CSC440 HW1 Mike Huber

Time Estimation

1. I expect to take about 8 hours total.
2. On the first night, I spent 1 hour on this homework.
3. On the other night, I spent 7 hours on this homework.

My Definition of Software Engineering

Software Engineering is the craft of creating software systems from the ground up. This starts with the idea, which leads to the design, which leads into the creation of the system. After this system is made, it is then finally modified and maintained to stay updated with the needs of the users.

Other Definitions of Software Engineering

1. "The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software" *The Bureau of Labor Statistics - IEEE*
2. "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" *IEEE - Standard Glossary of Software Engineering Terminology*
3. "An engineering discipline that is concerned with all aspects of software production" *Ian Sommerville - Software Engineering 8th Edition*
4. "The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines" *Fritz Bauer - Software Engineering, Information Processing*
5. "A branch of computer science that deals with the design, implementation, and maintenance of complex computer programs" *wwww.merriam-webster.com/dictionary/software%20engineering*

Introductions on SWEBOK Summaries

Chapter 1 - Software Requirements

The introduction of this chapter gives a clear definition of what it is going to explain, "... elicitation, analysis, specification, and validation of software requirements as well as the management of requirements during the whole life cycle of the sofware product." Terms are also set here to ensure the reader is understanding the terminology to be used. This chapter seems like it will have a more specific look at requirements, while the second chapter takes a high-level approach to the concept, as mentioned.

Sections: *technology and process*

Chapter 2 - Software Design

Detailing the way that the software will be set up and constructed is an important process, as highlighted in this chapter. Going from requirements to design takes a repitition of analyzing the blueprint produced from the design step, and modifying it until it meets the requirements set previously. This blueprint must be looked at both from a high-level perspective and a detailed perspective to ensure it has every part needed for the software at hand. This chapter actually does not deal with every part of design, but specifically D-design (decomposition design) and FP-design (family pattern design).

Sections: *design*

Chapter 3 - Software Construction

Construction of software is connected strongly to design and testing. A good deal of design is actually performed during the construction process. Unit testing and integration testing are both main activities during construction.

Sections: *technology*

Chapter 4 - Software Testing

Testing dynamically is a good strategy since the input on software can vary, and sometimes machine even react different upon the same input. However, there can be infinitely many input options, making exhaustive testing impossible. It is because of this that testing will have a finite set of tests to be run to cover as much as possible on the software. Risk analysis techniques are applied to determine these sets of tests. Software testing is done throughout the entire lifecycle of software.

Sections: *technology and human*

Chapter 5 - Software Maintenance

Since the development of software must meet user requirements, that software must adapt even after deployed in most cases. Any defects must be found and corrected. Maintenance can be done during both predelivery and postdelivery. Predelivery includes planning, maintainability, and logistics for transition. Postdelivery includes modification, training, and interfacing with a help desk.

Sections: *human and design*

Chapter 6 - Software Configuration Management

A formal definition of the discipline of identifying the configuration of a system at distinct points in time is the following

```
*A discipline applying technical and administrative
direction and surveillance to: identify
and document the functional and physical
characteristics of a configuration item,
control changes to those characteristics,
record and report change processing and
implementation status, and verify compliance
with specified requirements.*
```

Software configuration management relates closely to the quality assurance activity as well as being the supporting process that benefits almost everyone in the software life cycle.

Sections: *human*

Chapter 7 - Software Engineering Management

This part of management is applying activites to ensure everything runs smoothly. Some difficulties this management role may run into include the client not knowing what is feasible, changing conditions based on new understanding, changing requirements, and balancing the creativity and discipline that goes into software engineering. Along with management understanding at least the basics of what goes into software engineering, software engineers should understand at least the basics of the management role.

Sections: *human*

Chapter 8 - Software Engineering Process

Traditional engineering processes include taking a form of energy or physical entities and transforming them. Software engineering processes include developing, maintaining, and operating software. This particular chapter focuses mainly on planning, assurance, and control processes.

Sections: *design and technology*

Chapter 9 - Software Engineering Models and Methods

Model provide an option to problem solving. Methods provide an option for specifications, design, construction, and testing. Both models and methods vary greatly, from focusing on just one part of the software life cycle to looking at the entirety of the life cycle.

Sections: *process*

Chapter 10 - Software Quality

There are many definitions to quality, and a couple of different meanings as well. Each of which are fitting. Even individuals and companies have different definitions of the term. Generally though, most can come to an agreement on parts of each definition including meeting requirements accurately and satisfying specific conditions.

Sections: *design*

Chapter 11 - Software Engineering Professional Practice

This section of the guide pertains to the ability to work with responsibility and ethics when working with software. Professional practice usually is applied to areas with generally accepted bodies of knowledge and have codes of ethics and penalties for breaking those codes. These practices can apply to both technical and non-technical areas. A good example of codes for software comes from IEEE and the Association for Computing Machinery.

Sections: *human*

Chapter 12 - Software Engineering Economics

Business context is the idea of this section of the guide. Focusing on this, software has costs for the activities of the software life cycle but the finished product receives profit. The balancing of this is what software engineering economics is all about. Things that could be easy decisions from the technical side, could very well be difficult decisions from the business side. It's important for engineers to be aware of the business side of decisions to really understand the full viability of those decisions.

Sections: *process and design*

Chapter 13 - Computing Foundations

Software Engineering is based on both Computer Science and Mathematics, so this section is focused on the core principles of computing. However, it is important to not over exaggerate the impact that Computer Science has on Software Engineering. A good example that is made in the introduction of this chapter is computer graphics. While important to Computer Science, graphics don't play a role in Software Engineering.

Sections: *technology*

Chapter 14 - Mathematical Foundations

Logic is crucial to programming, and to learn that logic, software engineers use mathematics. That is what this chapter focuses on. There is no ambiguity to mathematics. Since this is the case, software engineers need to have the ability to have a precise abstraction on multiple domains but also have the rigid logic of mathematics.

Sections: *human*

Chapter 15 - Engineering Foundations

This chapter deals mainly with the knowledge and skills common to software engineering and include many different type of methods and techniques that are helpful to any software engineer. Using these methods and techniques will allow software engineers to develop and maintain software to the best of their ability.

Sections: *process and human*

Exerpt of The Mythical Man-Month Summary

Skipping this part as we plan to read the entire book instead of exerpt.

No Silver Bullet Summary

Introducing the idea of this "no silver bullet" idea is very interesting in this essay, as the author compares software to werewolves. He states that the legend of fighting werewolves was of course with silver. This was a quick and fairly simple solution to a horror. Although there are no silver bullets to quell the horror of software issues, the author reassures us that as long as we stay persistent with innovations, there will eventually come a solution.

There is a great comparison of softare to hardware. The author mentions that in human history no technology has grown as rapidly has hardware has. Software is merely attempting to keep up since there aren't many patterns or things that can be repeated over and over to result in progress. Buildings were another comparison made where there are aspects that are done in the exact same way each time they are done. Software has the difficulty of always being different, and if there are similarities in any way, they are most likely combined together to form one instead of multiple.

Essential difficulties of software are discussed fairly in depth. The ones mentioned in this essay are complexity, conformity, changeability, and invisible. Complexity is explained as essential versus accidental and that it is something natural to software because software engineering is working with something created by many people instead of just one being. Another interesting point the author makes is that software being invisible doesn't just mean the user cannot see what is going on behind the scenes. It means software engineers cannot make an abstract model of what they are designing or creating such as an architect can make blueprints and easily see where dimensions won't work and thing of that sort.

Growing software instead of building software is another part of this essay that has really grabbed my attention. Not to make this summary about me, but I have personally tried creating programs, albeit simple ones, in the manner that the author mentions in this essay and have had very positive outcomes. When the system itself is entirely working, though not fully fleshed out, it is a really good feeling. Then adding in functionality and ensuring requirements are met from the skeleton is much easier than if taken from a different approach than a top-down one.

PPT Slide Answers

1. Software Engineers must have the ability to *abstract*.
2. The name of the SE game is *dealing with complexity*.