Assignment for Lecture 1: Generic View

LAI Hui Shan
m5281022

1. What are three ways to deal with a wide variety of software functionality? Please explain each of them.

Ans:

First, **software engineers can specialize**. To some extent this has already happened. There are some software development firms, for example, that specialize in business and accounting software.

- A more common mode of specialization is in-house software development. A firm that uses software extensively will often have a department dedicated to software development.
- The software engineers that work in that department will automatically develop a specialization tailored to the needs of the firm.

The second option is that **people that are trained in the application area develop programming skills**.

- This is a workable option for relatively simple software.
- For more complex software, the loss in understanding of software construction may outweigh the advantage of familiarity with the application domain.

A more general solution to the problem is to modify the **software development process**.

- The basic idea is to allocate a significant amount of time early in the development on **Domain analysis**.
- Domain analysis involves communicating with experts in the application area, noting important concepts and their relationships.
- During domain analysis, software developers build a domain model that can be readily converted into software.
- Another important process change involves building the product in stages and getting feedback from the customer about the preliminary versions.

2. Indicate two aspects (dimensions) that define software complexity. If you have software being developed in your laboratory, analyze which quadrant it belongs to.

Ans:

Software complexity is defined by two dimensions: **technical complexity** and **management complexity**.

In my lab, I am currently developing a virtual reality educational tool, which likely has high technical complexity due to the need for advanced graphics rendering and user interaction design. However, as the project is relatively small, with only a few developers and straightforward management requirements, it falls into the category of **high technical complexity** but **low management complexity**.

3. What are the principles of software engineering that can be applied to all type of software system?

Ans:

Some fundamental principles apply to all types of software system, irrespective of the development techniques used:

1. Systems should be developed using a managed and understood development **process**. Of course, different processes are used for different types of software.
2. **Dependability** and **performance** are important for all types of system.
3. Understanding and **managing** the software specification and **requirements** (what the software should do) are important.
4. Where appropriate, you should **reuse** software that has already been developed rather than write new software.

4. What general issues make software development difficult?

Ans:

General issues that affect most software:

- Heterogeneity
    - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- Business and social change
    - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- Security and trust
    - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

5. What are the main four software process activities?

Ans:

Software process activities:

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

6. Please summarize "No Silver Bullet" by Fred Brooks within 300 words.

Ans:

Fred Brooks published his “**No Silver Bullet**” essay in IEEE Computer magazine in 1987. In this essay, Brooks described the "Silver Bullet" as a technology or practice that will magically and immediately double the productivity of programmers will not appear in the next 10 years.

Brooks suggests the following to address the complexity:

- Use the large market to avoid building what can be purchased.
- Use rapid prototyping as part of the development cycle plan when establishing software requirements.
- Grow the software organically (systematically), adding more and more functionality to the system as it is implemented, used, and tested.
- Identify and foster a younger generation of great concept designers.

7. Please summarize about “Software Crisis” within 300 words.

Ans:

Software engineering was spurred by the so-called software crisis of the 1960s, 1970s, and 1980s, which identified many of the problems of software development. Many software projects ran over budget and schedule. Some projects caused property damage. A few projects even caused loss of life. The software crisis was originally defined in terms of productivity but evolved to emphasize quality. Some used the term software crisis to refer to their inability to hire enough qualified programmers.

The following are examples related to the software crisis:

- **Cost and Budget Overruns**: This decade-long project from the 1960s eventually produced one of the most complex software systems at the time. OS/360 was one of the first large (1000 programmers) software projects. Fred Brooks claims in The Mythical Man Month that he made a multimillion-dollar mistake by not developing a coherent architecture before starting development.
- **Property Damage**: Software defects can cause property damage. Poor software security allows hackers to steal identities, costing time, money, and reputations.
- **Life and Death**: Software defects can kill. Some embedded systems used in radiotherapy machines failed so catastrophically that they administered lethal doses of radiation to patients. The most famous of these failures is the Therac-25 incident.

8. Please summarize what you personally think about the ethics and practices of software engineering that you will be strictly adhering to, within 300 words.

Ans:

I believe that being a software engineer involves more than just technical skills; it carries significant ethical responsibilities. To be respected as professionals, we must act with honesty and integrity, which goes beyond simply following the law—it’s about adhering to a set of moral principles.

In terms of professional responsibility, confidentiality is key. As engineers, we should always respect the privacy of our clients and employers, regardless of whether a formal agreement is in place. Competence is also important—it's crucial not to misrepresent our abilities and to only take on work within our expertise. Additionally, we must be mindful of intellectual property rights, ensuring we follow local laws and protect the intellectual property of our employers and clients. Finally, we need to avoid computer misuse. Using technical skills responsibly means refraining from unethical activities like spreading viruses or even misusing company resources for personal tasks.

9. Apart from the challenges of heterogeneity, business and social change, and trust and security, suggest other problems and challenges that software engineering is likely to face in the 21st century.

(Hint: Think about the environment.)

Ans:

In my view, software engineering in the 21st century will face several key challenges. One of the biggest is **sustainability**, as reducing the energy consumption of software systems will be crucial for minimizing environmental impact. I also believe that ethics will play an increasingly important role, especially with the rise of AI. We need to ensure fairness, protect privacy, and be socially responsible in the systems we create.

Another challenge will be managing **complex systems** like IoT, which will require innovative approaches to coordination and maintenance. Additionally, I think **globalization** will demand flexible software that works across different cultures and legal systems. Lastly, ensuring **long-term maintenance** of software will be critical to keeping systems secure and reliable over time. These are the challenges I see shaping the future of software engineering.

10. Explain how the universal use of the web has changed software systems and software systems engineering.

Ans:

The universal use of the web has led to the availability of software services and the possibility of developing highly distributed servicebased systems. Additionally, web-based systems development has led to important advances in programming languages and software reuse.