CS 410/560 - Software Engineering

Assignment 1 G CC

Group members: Yi Ren (002269013), Wentao Lu (002276355)

Introduction on Software Engineering

1. Define the term software engineering.

The definition of the term software engineering evolves, but I believe this term was proposed to solve the software crisis.

In the beginning, NATO conference defined that *Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.* So, this is an original definition that only concentrates on the effect of software, rather than the life cycle.

In the 1990s, IEEE gave the standard definition: *Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.* This definition mentioned not only the principles for software engineering but also some procedures of the modern software life cycle, such as development and maintenance.

Both definitions mentioned above introduce the engineer principles into the software to make every procedure in software more reliable, sound, and efficient.

[1, page 2-6]

2. What are the essential characteristics of software engineering?

1. Concern the development of large programs:
   The software nowadays is always completed by groups for the vast workload, because they need to satisfy the more complex requirements than usual. In this way, the scale of the modern program gets larger and larger, since developers need to invoke many components from other developers.

2. The central theme is mastering complexity:
   With the larger scale of programs, the complexity has become an issue for software engineering. Rather than the architecture of software, there will always be a large number of details.

3. Software evolves:
   The world today is always changing very quickly; as a result, the software must also evolve to stay current with the development trend. In such a rapidly changing industry, it comes at no surprise that things can soon become out-of-date. For example, we always see that the requirement varies after delivery due to budget concerns, market revolution, etc. Sure enough, the software has to evolve accordingly as well.

4. The efficiency of software development is essential:
   Now our world has witnessed a constantly growing demand for new software applications, but the supply side, such as workforce resources are falling short. To deal with this challenge, we must seek a much more efficient way of software development and maintenance, which guarantees to ensure fast delivery under constraints, software reusability, robustness, and so on. Consequently, developers must use better methods and tools to work on software more efficiently.

5. Regular cooperation between people:
   Given the scale and complexity of many large projects, often times, many people now have to work together on the same software or problem. It's very common for people to work with others in a different city or even country, such as in the case of an open-source project. For example, I used to work for mobile phone manufacture, where projects tend to be extremely large which requires hundreds of developers to work together. However, while our application developers are in the US, system groups are in Korea, the camera group is in India, and the testing group is in China. All the cooperation is arranged online with certain tools.

6. Support users effectively:
   To meet the needs and expectations of users and build user-friendly software, we need to know about the users' requirement in detail and support them properly. We must pay attention to users' requirement and feedback, and thoroughly understand their practical concerns so that reliable and robust software will be developed.

7. Software engineering is a field in which members of one culture create artifacts for members of another culture:
   Software engineering usually involves members from different areas. For example, a back-end developer who has a solid foundation in C++ might not be an expert in web design, a software engineer who has strong technical skills might lack the domain expertise in management or marketing. To better cooperate as a team, I think the product manager should provide the specific requirement, it's also necessary to sometimes learn the basics in another field as well.

8. Software engineering is a balancing act:
   Due to the fuzziness of reality, user requirements and specifications are not set in stone. Sometimes, requirements or the designed functions of software is, in fact, mutable and negotiable, which may change for many reasons such as technical constraints, policy limit, or can be carried out in another manner. We should bear in mind that this balancing act is complicated, which always involves some trade-offs.

[1, page 6-8]

3. What are the major phases of a software development project?

The major phases include *requirement engineering, design, implementation, testing, and maintenance*. All of these phases could be arranged as a linear life cycle, but there are also some nonlinear models where some phases will overlap.

- Requirement engineering
  In this phase, we need to make a description of the problem that needs to be solved, which

include the function of the software, some specification documents which specify some performance requirements, such as response time, capacity, and so on. The result of this phase: requirements specification

- Design
  In this phase, we need to split the problem into high-level components, and then specify the function of each component and the interfaces between components. The result of this phase: technical specification

- Implementation
  In this phase, we start to implement every component in detail with the requirements specification and technical specification. If the size of the executable code is huge, the use of pseudocode can be very helpful. The result of this phase: executable program

- Testing
  This phase is concerned with testing the correctness of software; it could even start earlier than implementation. Most importantly, validation checks if we are building the right system, and verification checks if we are building the system correctly. There are various kinds of testing, such as unit testing, integration testing, system testing, black-box testing, white-box testing, pressure testing, and so on. The earlier that errors are detected, the cheaper it is to correct them.

- Maintenance
  This is the phase after delivery; in this phase, we need to repair some errors, complete some adaptive maintenance, and also try to increase the performance of the system.

[1, page 12-14]

4. What is the difference between verification and validation?

- Verification:
  To test if the transition between two phases is correct, in other words, to verify if we are building the system right.

- Validation:
  To test if the system fulfills the requirements from users, or to say if we are building the right system.

[1, page 13-14]

5. Define four kinds of maintenance activities.

- Corrective maintenance:
  There will remain some errors after testing when the software is delivered; maintainers should repair these errors as soon as possible.

- Adaptive maintenance:
  Sometimes the software needs to be adapted to a different environment; maintainers need to adapt the program from one environment to another. For example, from Windows to Linux, from PC to mobile phone.

- Perfective maintenance:
  The requirements may change after delivery; as a result, some new features or functionality could be required to be added to the software. Besides, maintainers also need to perfect their system, such as improve the system performance and user experience.

- Preventive maintenance:
  Maintainability is important for software, one with high maintainability means it cost less in the maintenance phase. Maintainers need to work on preventive maintenance, such as updating their documents and adding comments in the code to improve maintainability.

[1, page 16]

6. Why is the documentation of a software project important?

There are many kinds of documents in the whole process of software development, delivery and maintenance. Maybe most of the developers hate to write documents; however, you can not deny the importance of high-quality documents.

For product managers, they need the project plan to convince the customers.

For developers, they need the requirement specification and architecture description to develop every component of the system.

For testers, they need the test plan to complete the testing work of each phase.

For maintainers, in most cases, they need the interface description and maintenance documents to learn about the project.

For users, they need user documents such as user manual to understand the usage and picture of the software better.

[1, page 14-15]

7. Explain the 40–20–40 rule of thumb in software engineering.

That means, in most cases, 40% of effort should be spent on requirement engineering, another 40% on testing, and only 20% on coding.

However, this effort distribution is not set in stone; in essence, it merely implies that we don't need to spend too much time on implementation. In practice, it's better to spend more effort on the requirement engineering phase and design. This way, we can detect and correct errors at an earlier time with much less cost. *The longer you postpone coding, the earlier you are finished.* In other words, if we just rush into coding too early without working carefully on the previous phases, it takes even longer to finish the project.

[1, page 15]

8. What is the difference between software development and software maintenance?

Software development is a set of phases before delivery, while software maintenance is after delivery. Software development is a process that includes requirement engineering, designing, implementation and testing. It's about the transition from requirements to product. On the other hand, software maintenance is all about work after software delivery, this may include some coding work such as adding new features and fixing some errors, but the main purpose of software maintenance is to increase the maintainability, performance and quality of software, in other words, make software easier to maintain and use.

Software Management

1. In what sense is the phrase software development project a misnomer?

The phrase software development project is not just about development; the project refers to the entire system, which includes software, hardware, documents, running environment, people who use the software, procedures that govern the software and so on. The notion *software development project* should be understood in the broader sense.

[1, page 35-36]

2. What are the major constituents of a project plan?

1. Introduction
2. Process model
3. Organization of the project
4. Standards, guidelines, procedures
5. Management activities
6. Risks
7. Staffing
8. Methods and technique
9. Quality assurance
10. Work package
11. Resources
12. Budget and schedule
13. Changes
14. Delivery

[1, page 38-40]

3. List five dimensions along which a software development project has to be controlled.

Time, Information, organization, quality, and money. [1, page 40]

4. How may software development become a capital-intensive activity, rather than a labor-intensive one?

From labor-intensive activity to capital-intensive activity, the critical point is to increase productivity and reduce cost. Two methods are mentioned:

- Use better tools.
- Use software rather than build it yourself.

[1, page 42]

The Software Life Cycle Revisited

1. Describe the waterfall model of software development.

The waterfall model can be illustrated as a graph below, it is a very simple model of which the phases are arranged sequentially, verification and validation (V & V) are required in every phase. Despite that software engineering is full of compromises, this model captures to some degree how a project should be run from a very straightforward perspective.

- Advantages: This model includes iteration and feedback between different phases. Since each phase has verification and validation, so requirements can be fixed as early as possible. For example, if a problem is found at design phase, we need to go back to this phase and fix all the issues. Such feedback can effectively reduce uncertainty and cost.
- Disadvantages: The assumption that phases are arranged in a sequence is relatively rigid. Often times, it's hard to comply with this model because it's not always realistic. In practice, different phases may overlap, and developers may need to jump between some phases without following the sequence. For example, not all of the design effort is spent in the design phase; when an issue is found later, some design work may also be included in the coding and testing phase.

[1, page 48-49]

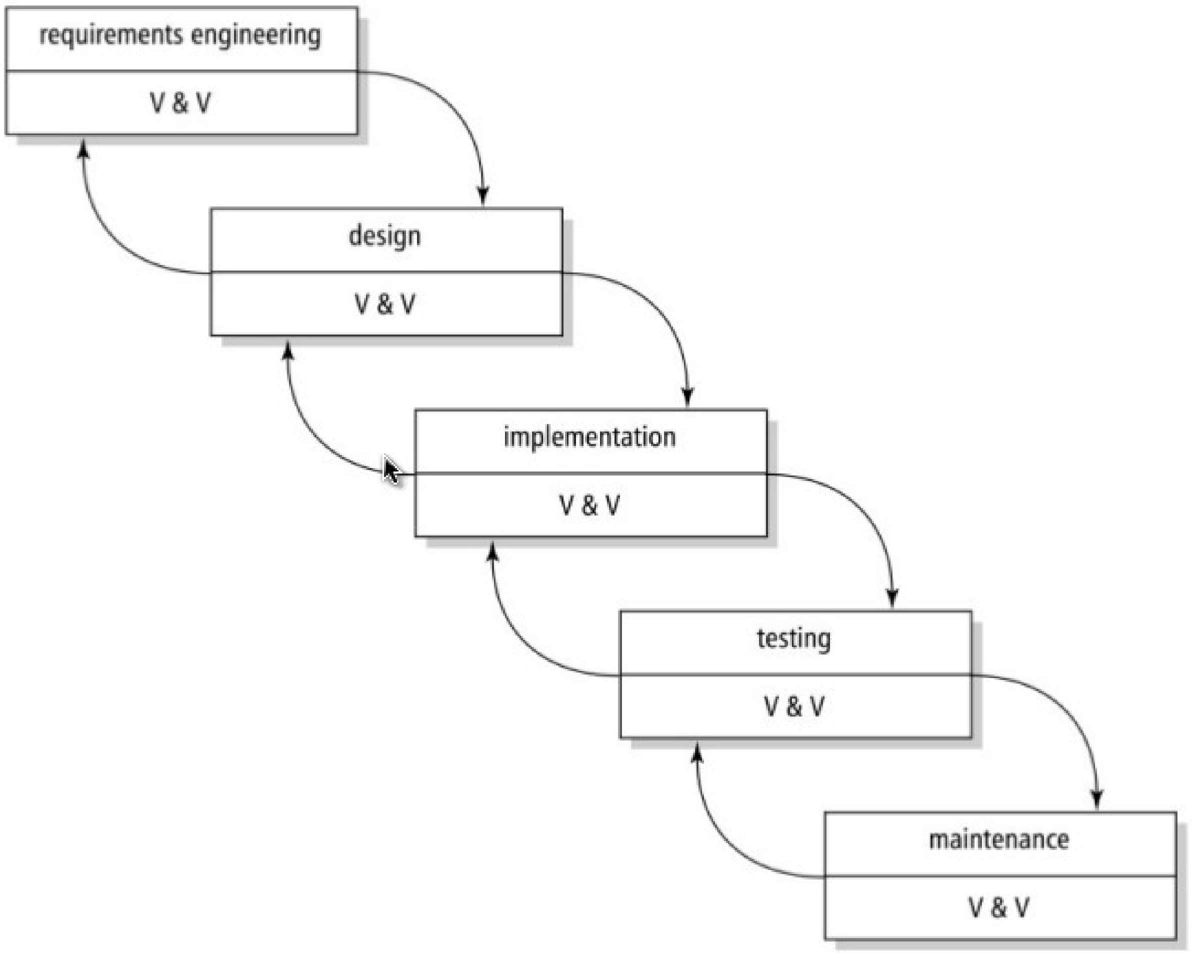

Figure 1: waterfall

2. Describe the v-model of software development.

The V-model is evolved from the waterfall model, but it is more flexible. In a V-model, the sequence of phases forms a V-shape instead of a single straight line, so different phases can be associated with each other. For example, testing may start well before the coding phase. After requirements engineering finishes, testers can immediately begin to deal with acceptance testing.

- Advantages: Problems and errors could be found earlier than the waterfall model.
- Disadvantages: When requirements are changed, there will be more work to do compared with other models, because testing phases also need to restart accordingly.

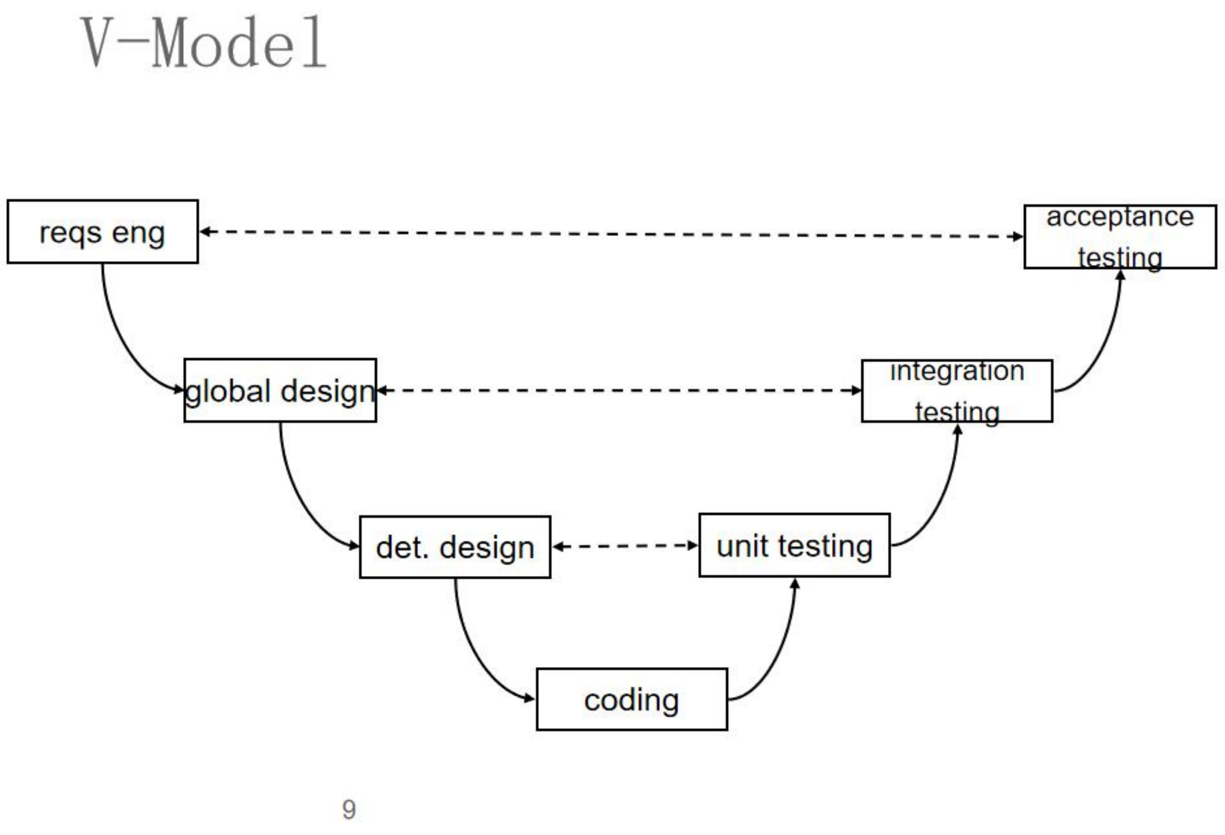

Figure 2: v-shape

[2, page 9]

3. Describe the Rapid Application Development (RAD) approach to software development.

RAD, rapid application development, is also one of the iterative development process models. RAD is consists of four phases: *requirements planning, user design, construction and cutover*. In the RAD model, the deadline is fixed and immovable; thus, some of the functionality may need to be removed temporarily. To deal with the priority issue, all of the requirements are divided into four levels: *Must have, should have, could have and won't have*. Requirements of high priority will be constructed prior to those of low priority. For example, must haves requirements will absolutely be constructed, should haves and could haves requirements may be implemented if the developer has enough time. Won't haves requirements will not be coded up until the next iteration.

[1, page 57-59]

4. Discuss the main differences between prototyping and incremental development.

For prototyping, we need to implement a simple system or a demo which has only some of the functionality. The demo is used to communicate with the customer to better accommodate their needs, elicit their feedback, and better understand their requirements in detail. Given this simple purpose it aims to serve, the quality of the product is not required, and the final delivery is just a demo. The resulting system is easier to use, but there could be only a few features.

For incremental development, the software is developed and delivered in small increments, that is, in a step-by-step manner. During each iteration, the waterfall model is applied, and users are actively getting involved in directing iterations.

The main difference between them is that they serve different purposes. Prototyping is applied when we need to clarify users' requirements and ensure that we are building the right system, the final delivery is not necessarily based on the first. If something goes wrong in the middle, the first delivery could be discarded. On the other hand, incremental development is mostly concerned with cost and risks, it is applied in order to avoid a Big Bang effect; that is, we don't want to find a severe problem only at the last minute. The term increment implies that the development evolves over time, so the final delivery is always based on the first delivery.

[1, page 51-57]

5. Discuss the main differences between incremental development and RUP.

Incremental development is one of the agile methods, while RUP (Rational Unified Process) lies somewhere in between agile methods and document-driven methods. RUP is an iterative development process, it has well-defined process, but it also has iterations in every phase.

[1, page 64-65]

6. Discuss the law of continuing change.

The law of continuing change is about software evolution, it reveals the fact that software will always face with new changes after delivery, it is continuously evolving until people find it too inefficient or too costly to maintain any further. When that happens, it's better to restructure or rebuild the whole system.

[1, page 66]

7. How does the spiral model subsume prototyping, incremental development, and the waterfall model?

The spiral model can subsume different models because it has risk analysis in every phase and can be adjusted under different conditions.

Prototyping model: If the requirements are seen as risky, people can follow the spiral a few times until they solve the problem. However, if the requirements are precise, it suffices just to follow the spiral once.

Incremental development: Every increment can be seen as one track of the spiral.

Waterfall model: Every phase of the spiral model can be seen as a waterfall.

[1, page 56-57]

8. Explain the principle of XP, and the practices pair programming and refactoring.

XP stands for Extreme programming, it's absolutely a pure agile method because XP mainly focuses on the implementation speed. There are many practices that XP takes to increase efficiency, pair programming and refactoring are two of those practices.

Pair programming: two programmers work on the same machine, when one of them is coding, the other will look at the codes and give some advice. This practice can increase code quality and solve the problem more efficiently.

Refactoring: developers change the structure of the system to improve the code quality, for example, from MVC to MVP, but the system behavior stays unchanged.

[1, page 62-63]

9. What is a software product line?

A software product line is a set of software systems that share elements. The key point is to reuse items, such as code, design or architecture by plan, rather than by accident.

[1, page 70]

10. What is the main purpose of having an explicit description of the software development process in a process model?

1. For communication purpose:
   that means a precise description of the software development process will make it easier for people to communicate with their colleagues, without too much ambiguity.
2. For management purpose:
   the precise description helps the project manager to assign and track tasks in a more efficient way.
3. For automated support:
   since automated tools are widely used in modern software development, an accurate description lays a foundation for automated support, which makes it possible for machines to complete many tasks automatically.

[1, page 73-74]

11. What is process enactment?

Process enactment means the execution of the process by either humans or machines.

[1, page 74]

12. Discuss the key values of the agile movement.

Four key values ensure the efficiency of agile methods.

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

[1, page 50]

References

1. Hans van Vliet. Software Engineering: Principles and Practice. Wiley 2007.

2. The Software Life Cycle Revisited.ppt