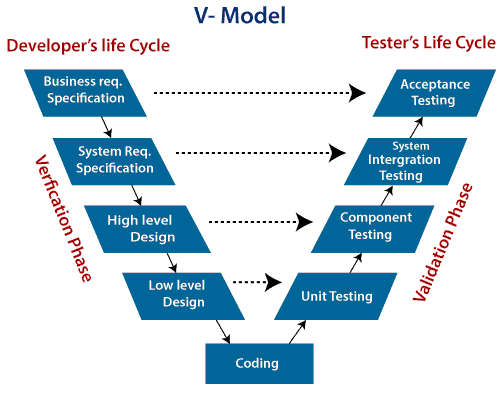
**Weekly Assessment 1**

**CSET PM session Fall 2022**

**30 points**

**Kaedyn Kessler**

**Answer the following questions -**

1. Is software manufactured, engineered, or both? What’s the difference?   
   In my opinion, software is engineered rather than manufactured. My reason for this lies in the manner in which software projects are completed; majority of the time, software projects are never truly finished, as updates and bug fixes are often released to the public proceeding release.
2. How likely is software to fail as time passes? You can use a graph to answer.  
   Software failure typically occurs due to a fault within the development team. Poor communication, inability to overcome a particular challenge, and a general lack of planning are all reasons why a software may wind up failing over time. The chances of this happening, in my opinion, would likely decrease over time. The initial public release and the feedback of said release sets a general precedent among the public.
3. Why must software change?  
   Software must be able to change and adapt to consumer needs and wants, which are always in a state of flux. Any inability to follow through with such adaptations would most definitely result in a failure of the software.
4. How would you define software engineering?  
   In my opinion, software engineering can easily be defined as a practice of software development in which a certain plan or methodology is followed to deliver fully functioning software.
5. What is the difference between -
   1. Software engineering and system engineering?  
      System engineering refers to the process of developing the software that will allow system hardware to be interacted with via software that is to be manipulated by the user. Naturally, software engineering refers to the process of building that user software.
   2. Software engineering and computer science?  
      Computer science as a definition is best described as an umbrella term that a subject such as software engineering would fall under. CompSci refers to a wider range of subjects within the computing industry, including information technologies and hardware, as well as software and application building. Software engineering, however, specifically refers to the development/construction of computer software.
6. Describe David Hooker’s general principles
   1. The Reason it All Exists: the core values of why a software is being created should always be kept in mind during development, as it’s purpose is essential upon production delivery
   2. Keep It Simple, Stupid! (K.I.S.S.): the design and engineering of software needs no unnecessary complication, but should rather be kept as simple as possible without being detrimental to the end-result.
   3. Maintain the Vision: similarly to the first principle, a clear and concise concept for a software in development should be kept in mind at all times. Knowing what the product is supposed to look like and how it will function is of upmost importance.
   4. What You Produce, Others Will Consume: one key component of a software development is the assurance of it’s accessibility among its intended consumers.
   5. Be Open to the Future: well-developed software should be able to stretch its longevity as much as possibility. Assuring that there is leeway for change in the future is extremely important.
   6. Plan Ahead for Reuse: a development team should plan to use previously-developed code for later components in order to save time and energy that would be much better spent elsewhere.
   7. Think!: the final and arguably most simple principle states that developers must fully think through each critical decision before putting it into action.
7. Are the facts or fiction of the following statement? If fiction, then provide the correct facts -
   1. Once the developers deliver their software product; their job is done.   
      Fiction: one would likely argue that a development team’s hardest jobs begins upon delivery of a software production; user feedback, including bug reports and recommendations for improvement, is vitally important for developers to take into consideration. Otherwise the product might not survive very long.
   2. Planning is not necessary for software engineering. The best practice is to dive into coding right away.  
      Fiction: in the majority of instances, having at least a general plan to follow is extremely important in software engineering. The chances for the end-result to be a failure is very likely otherwise.
   3. Modern software is very flexible. Hence it is not very difficult to incorporate change into them.   
      Fact.
8. What is a software process model?  
   A software process model can be described as a methodology that a development team would follow to deliver the best software possible. What model they follow depends on the parameters of the project itself.
9. List and describe the stages of a prescriptive model.
   1. Communication: I would ask the school what sort of features they were expecting to see in the program, such as information they wanted to be displayed for each student, the login requirements for staff access, and more.
   2. Planning: I’d work with my team to assign specific tasks to each member, as well as drawing out a map in which everyone can follow. I would make sure everyone was on the same page at all times, and that all the features requested by the school are planned to be implemented. Furthermore, I’d work with everyone to create a schedule to ensure that all members have the proper time to complete their assigned work.
   3. Modeling: I’d work with the team to develop a detailed model to focus on how the application is going to look visually, as well as how it will perform the required tasks.
   4. Construction: Once all the prerequisites have been completed, the team will then begin to work on the programming. Everyone will focus on their roles and collaborate together to make sure all features are compatible and functioning.
   5. Deployment: upon completion, the software will be sent to the school for testing and feedback. Any bugs will be corrected and fixed, and any features will be optimized upon request.
10. Describe the waterfall model in detail. In which cases would you prefer the waterfall model, and in which cases would you not?  
    The waterfall model is an approach to software development that follows a concrete set of steps based upon prerequisites that must be met and achieved. Each step followed must go through an approval process before the following step can be approached, hence the name, “waterfall.” While this process is a well-known classic method to software development, it is not very helpful when creativity and uniformity within the development team should be an essential. Not only that, but this process can wind up being costly. However, in cases where there are requirements predefined by the client, then the waterfall method could be considered.
11. Describe the V model with a diagram. How is it an improvement over the waterfall model?  
    The V Model can be considered to be an improvement over the waterfall model when looking at its flexibility. In comparison to the waterfall model, the V Model allows for much better future adaptions that could be potentially requested by the client. Overall, the V Model allows for a greater chance of success and allows for many required activities to be done simultaneously.
12. Describe the stages of planning as shown in the V model.  
    Within the V-model, there are two phases/cycles that follow their own stages of planning, called the design phase and the testing phase. Within the design phase, developers must go through requirement analysis, system design, architectural design, and module design. On the other hand, the testing phase includes unit testing, integration testing, system testing, and user acceptance testing.
13. What are *high-level design* and *low-level design*?   
    The difference between high-level and low-level design are simple in definition; high-level design refers to the general design of a system’s architecture and its interaction with other essential primary components. Low-level design, on the other hand, can be best described as the design of individual software components within a large software system.
14. Describe the advantages and disadvantages of the incremental model.   
    While the incremental model is useful in adapting to change during the development process, there is a downside in time management. To determine if this model is applicable to a specific project, a development team must assure that there is a wide enough time frame to fulfill all necessary requirements.
15. What is an evolutionary process model? Name two evolutionary process models and explain their advantages and disadvantages.   
    The evolutionary process model can be described as a combination of both the iterative waterfall model as well as the incremental model, with the purpose of delivering a basic software product in which features are added to as time goes on. While both these models offer the advantage of assuring that each component of a software functions at its best, time can be an ultimate cost.
16. What is agile? How can a development team be agile?   
    Agile is a type of software development that revolves around more than just frameworks, models, or practices. Agile is a type of software development approach that focuses on the ability to adapt to change, as well as the interaction between the working developers and how that interaction takes place. Self-organization is of extreme importance with agile development. Customer requirements are not compatible with agile as well, as contract negotiation is not a key value.
17. State and describe the agile values.
    1. The early delivery of software to the client followed by consistent updates and added features is top priority.
    2. Adaptability is extremely important, with a change in requirements even late in development being completely acceptable.
    3. Release function software over a preferably short period of time, ranging from a few weeks to a few months.
    4. Cooperation and collaboration among developers and other business workers is a daily essential during project development.
    5. Developers of a project must be motivated and trusted to complete the work they were assigned, and should be accommodated with any and all resources they require.
    6. Communication across the development team is best done via face-to-face conversation.
    7. A software’s current functionality is the best measurement of its development progress and resulted functionality.
    8. A steady, unchanging pace in development speed is extremely important in the Agile development process; any changes should not cause significant change in such.
    9. Focus on good design and technical quality should remain constant.
    10. Simplicity is crucial; any work needing development should be maximized at all times.
    11. Self-organized development teams output the best results in software quality.
    12. Self-reflection within the team should occur regularly. Any and all behaviors needing change should be acted upon immediately.
18. What is XP (extreme programming)? State its advantages and disadvantages.  
    Extreme programming (XP) is a framework labeled under the Agile software development umbrella. The processes under XP involve changing requirements and the ability to adapt to the risks provided by deadlines. This framework isn’t utilized in most software development scenarios, but can still be extremely useful.
19. State five agile principles that you think are the most important. Explain your choice.  
    I believe that the five most important principles of Agile are adaptability, cooperation and collaboration among the development team, developer motivation and trust, self-reflection, and the early delivery of the product are most important. For me, the most crucial element of a developing software is the team that works on it, and how each and every developer interacts with one another. Without that healthy work environment, a well-designed software is simply impossible to create.
20. State five XP practices that you think are the most important. Explain your choice.  
    In my opinion, the five most important practices of XP are as follows: peer programming, the planning game, simple design, testing, and continuous integration. One of my personal favorite components of XP is indeed peer programming. Having two developers focus on the same piece of code allows for far better end results. Furthermore, it is very easy to fall into a trap of making things more complicated than necessary, and focusing on a simple design, especially if previously planned, is extremely useful.

**Good luck!**