Pledge: *I pledge my honor that I have abided by the Stevens Honor System.* - **Eric Altenburg**

1.6: As software becomes more pervasive, risks to the public (due to faulty programs) become an increasingly significant concern. Develop a doomsday by realistic scenario in which the failure of a computer program could do great harm, either economic or human.

A realistic doomsday scenario in which the failure of a computer program could do great harm would be applicable to the Coronavirus outbreaks that have been happening recently. Software is at the center of diagnosing illnesses and monitoring medical procedures, and if one of these software systems were to malfunction, then there can be serious repercussions such as injury and even death. Especially nowadays, outbreaks of the 2019 Novel Coronavirus (COVID-19), the ability to accurately diagnose an individual is extremely important to prevent the continuing spread of this disease. Another real-world example would be if open heart surgery were occurring and then the machine that is used for monitoring the patient's current health were to malfunction, then the patient's health and well being would be at risk.

2.8: Is it possible to combine process models? If so, provide an example.

Yes, it is possible to combine process models, in fact, some software development departments/-companies do not use a traditional process model. Instead, they end up using a proprietary process model which is just a combination of other more traditional process models that better suits their work flow/products.

A few examples of combined process models include:

1. Evolutionary process model
    - Combination of iterative and incremental approach.
    - Over time, incrementally create a more complete version of software (more so than the last) and for each of these incremental builds, a complete cycle of activities are completed.
2. Spiral model
    - Combination of iterative and sequential linear approach; waterfall with emphasis on risk analysis.
    - Similar to the evolutionary process model, it produces versions of software over time more complete and refined than the last.
3. Incremental process model
    - Combination of one or more waterfall models.
    - This model produces a series of releases that provide more functionality for the customer and these builds are individually designed, tested, and delivered at specific deadlines.

2.9: What are the advantages and disadvantages to developing software in which quality is "good enough"? That is, what happens when we emphasize development speed over product quality.

(Advantages)

3.2: Describe agility (for software projects) in your own words.

The agility software process is designed to deliver a product in a fast manner, and for the software to be delivered per the customers' specifications. If the customer decides to change their requirements, then the developers will be able to adapt and accept the challenges the new requirements may bring. The overall goals of agility mean that the software must be kept as simplistic as possible in that, only the requirements are to be met; nothing extra which helps with scope creep. Agile software development is also more growth oriented as well meaning that although it will have to adapt to new challenges and requirements, it will do so incrementally to avoid drastically disrupting the pace of the project and to make certain that any progress made will always be forward and no backwards. This also allows for the cost of the new changes to be minimal as everything is controlled and introduced in incremental steps.

5.1: Based on your personal observations of people who are excellent software developers, name three personality traits that appear to be common among them.

The top three personality traits among the best software engineers are:

1. Technical skills
    - The best software developers must have the level of technical skills required to be able to fully understand the legacy system in which they are working with, or possibly a client's system. Then figure out how to make improvements to the system, or propose a new one that will prove to be better than the previous. This individual will also have to figure out how to implement the new system with the older one.
2. Ethical skills
    - The morality and ethical skills an software developer must have is crucial as they may have access to personal details of a company that, if put in the wrong hands, can ruin said company or business. Specifically nowadays with AI, the developer must be aware of how they are coding up a program or application and how it can potentially be used.
3. Interpersonal skills
    - In most cases, software developers will be working in a team or managing a team, and to do so efficiently they must be able to properly work with individuals and express their thoughts effectively. For those working in upper management, they must properly delegate individuals to do tasks while being fair to the other team members in terms of work distribution and existing personal issues.

6.6: Of the eight core principles that guide process (discussed in Section 6.1.1), what do you believe is more important?

The eight core principles that guide processes—be agile, focus on quality at every step, be ready to adapt, build an effective team, establish mechanisms for communication and coordination, manage change, assess risk, and create work products that provide value for others—are all important under their own respective circumstances, however, I feel as though that building an effective team should be placed above the rest for a few reasons. When developing software, the individuals someone is paired with for a project are the people they will be interacting with the most; therefore, it is crucial that these individuals get along. Mutual respect and trust are a requirement among team members as the entire base to a team is build on this, without it, little to no work will get done due to differing views or the way processes need to be handled. No one can trust each other to assign tasks and get work done, and this can quickly derail projects. Each member must also be diverse and have their own respective skill-sets because too many assertive members can result in lots of clashing opinions, and no work will get done. On the other hand, too many passive individuals can lead to nothing being done or potentially the wrong thing being done; no one will speak up or suggest a different approach to an issue. It is for these reasons that picking a team with a diverse skill set and the potential for lots of respect and trust to be built be considered for the utmost priority over the other seven core principles.

7.1: Why is it that many software developers don't pay enough attention to requirements engineering? Are there ever circumstances where you can skip it?

7.5a: Develop a complete use case for making a withdrawal from an ATM.

8.1: Is it possible to begin coding immediately after a requirements model has been created? Explain your answer, and then argue the counterpoint.

8.10: How does a sequence diagram differ from a state diagram? How are they similar?

(Difference)

In terms of how the system behaves, the sequence diagram will give a detailed description of how the classes will move from state to state, whereas the state diagram will not have any information about classes. Also, the state diagram will explain how a singular class will change states based on external factors, however, the sequence diagram will show the behavior of the software with respect to time.

(Similarity)

They are both used to model the behavior of the software, just done so in different manners.