Christopher Morcom

CS 487

10 August 2019

**Discussion Questions**

**1.How has software development improved as a result of the “software crisis” discussed in lecture?, how is it still the same?**

Software development as a result of the “software crisis” has changed since there are different techniques which are used when creating complex software. For instance, software is broken down into smaller pieces and implemented in a way where pieces of software are reusable. Reusability is a feature built into software when “software pieces” are created in a way that they are flexible in an instance of maintenance. However, software development is still the same as today in the sense of how it is designed. Newer updates and versions, or even methods/libraries are always implemented due to constant changes in technology today, which is not too different from the “software crisis” in the past.

**2.Discuss benefits of the Waterfall approach as well as shortcomings of the Agile approach.**

The Waterfall Approach is a process that makes each consecutive step validate, reducing the chance of mistakes, which would in turn yield a higher success rate. The drawbacks to this approach are that each step is tedious and takes lots of time to complete in an acceptable way. Systems generated using the waterfall approach go through many iterations, making the other drawbacks and benefits yield more weight.

Unlike the Waterfall Model, the AGILE Approach has a high failure rate, where software may be lacking in several ways. AGILE also does not require validation periodically during development, causing errors in earlier stages of development to propagate into the prototype. However, some benefits of AGILE are that it is less complex and easy to build, and it tends to get the “big picture” or overall idea done with significantly less cost and effort than that of the waterfall model.

**3.Discuss an example of non-functional requirements which require design trade-offs in order to each be satisfied.**

Security and ease of use are examples of non-functional system requirements that may require design trade-offs to be satisfied. For instance, in my project, I created a system that requires user validation and content moderation. This system makes it harder for users to interact with the system because they have an extra step to take before using the system and will have to log in after set periods of time to achieve a secure, validated account. Or, their interests may be curbed while their content is moderated, which would detract from the “ease of use” idea..

**4.Why are software systems particularly well-suited for automation?, under what circumstances is it difficult create a software system which automates?**

In software engineering, all systems are built in a way that automates, or reduces work for a task. For instance, research is semi-automated since people can just look things up on google instead of indexing through a library by call number manually to find a book. Systems that require repetitive tasks are a good example of why software systems are well-suited for automation. However, some cases where an automated system is difficult to create is where there may be “vagueness” or “randomness” in results. One example is if we create a software system that moderates data. The system will have to bring subjectivity and context into the equation which is a hard problem to prove as solvable.

**5.Why does a compiler consistently translate source code accurately while humans frequently misunderstand each other?**

Source code compilation is ***absolute***, that is, the compiler will always create the same output for the same input. A compiler does not require context and does not know intention, making the compiler do exactly what it is told – even if the output is not the desired solution.

On the other hand, humans constantly misunderstand each other since we have context and intention. People tend to subjectively understand words and grammar, which can have many meanings, or change the meaning of a sentence. Since what we say is not ***absolute***, humans will misunderstand what is said. A common example of this is the Bible, where interpretations of it have changed over hundreds of years and various texts have been written.

**6.Why is it difficult if not impossible to prove perfection?**

Proving software perfection is a subjective proof that depends on the customer. For instance, in the iterative approach, we work through steps to achieve the goals that the customer set. The engineer’s perspective is that the number of iterations is irrelevant, but the goal is to satisfy the customer and ***reduce*** the need for later maintenance in order to save time and money. The software can’t be proven perfect since it is not possible to prove that the system can not be optimized or improved any more, or that there is a finite number of iterations.

**7.Discuss the concept of defensive design and give an example.**

The defensive design approach assumes that at users are not going to always use the software in a way that does not generate an error. From there, the system is built in a way such that it can handle any errors in a safe way that could recover the system so it would continue to work correctly. For instance, the Automated Instructor example from class and the homework would require a defensive design since users may ask the system questions at any time, make mistakes, etc. The data read from the sensors must be interpreted by the system and handled accordingly. If a student passes, move on to the next section, else, go back and review the material.

**8.Discuss the benefits of reuse and steps that should be taken to make components reusable.**

The benefits of software reuse stem from the idea that existing software is proven to work. To make components reusable, modules have to be generic such that they are not dependent on the system. The software also requires infrastructure and process to become modular such that it can be ported to other systems. Benefits include the following:

* ***Speed:*** Software may perform better than an untested, new system. Time spent developing the system is also greatly reduced.
* ***Peace of Mind:*** Software is proven to work correctly, so it becomes less complex to find and handle errors.
* ***Reliability:*** Since the software is known to work, it is reliable in the sense that it will always generate the same outputs for the same inputs (no randomness).
* ***Reduced Risk:*** Since the software is reliable, there is a lower risk that the reused software will break the system if the system is created correctly
* ***Ease of Maintenance:*** The reused software, if maintained, will be updated in a way such that it will always work better, which will reduce maintenance complexity for the entire system.

**9.Discuss differences and similarities between H-C-I and C-C-I designs.**

Human Computer Interaction (HCI) and Computer Computer Interaction (CCI) are similar in the sense that two entities need to communicate information which interpreted in an objective, or raw form. HCI is generally a means for humans to communicate through a system, while CCI communicates systems with a smaller chance of failure due to computer error. HCI requires that a human modulate their information such that the computer that the human interacts with will understand the input ***correctly*** in an objective and lossless way. CCI requires that the information passed between systems is done correctly, which can be done if the system is proven to work. HCI requires that the system works and that the human does everything correctly, while CCI only requires that the system works.

**10.What about “mission critical” makes development more difficult?**

“Mission Critical” makes software development extremely difficult since there must be no chance of errors occurring in a stringent defensive design. “Mission Critical” Software is a term attributed to expensive or dangerous scenarios where failure is not an option. For instance, the Ariane 5 was a failed system since the software was not designed to handle inputs greater than that of the Ariane 4, which caused the mission to fail due to software error.