CS347 "I pledge my honor that I have abided by the Stevens Honor System."

Julia Nelson Homework 1 03/07/2020

⇒ Answer all questions in a document. Be clear and describe your answer well and convincing. Most answers require several thoughtful sentences to convey the response.

⇒ Use MS, PDF, or other popular formats. Upload your answer in this Canvas Homework. Questions are from problems at the end of Chapters and numbered as in the 9th Edition.

⇒ This is an individual Homework assignment. Teamwork would constitute Stevens violation of Honor Code. No late homework will be accepted unless you have a valid excuse.

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

Use case: Usage of nuclear power plant coolant-software (faults)
Primary actor: Reactor Control System
Goal in context: Control system retrieves reactor's temperature from sensors, and then communicates with Coolant-software to release the appropriate amount of coolant into the reactor.
Precondition: The reactor and control system are operating in normal and safe conditions.
Trigger: Temperature in the nuclear reactor begin to increase.

1. Control system requests reactor's temperature from sensor.
2. Control system reads the temperature.
3. Control system logs the temperature.
4. Control system communicates with coolant-software to release a specific amount of coolant.
5. Coolant-Software does not release any coolant. *Faults*
6. Temperature in reactor is not decreased and continue to increase.
7. High temperatures cause explosion and meltdown in.
8. Large surrounding areas and people exposed to harmful radiation.

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

Process models are common structures used to help organize and efficiently complete a project. While various process models contain similar aspects to others, their generic frameworks aid more specifically towards different types of projects. However, it is possible for a software development team to combine different process models in order to better suit their goal.

An example of a design process combination is the joinder of the Waterfall and Prototyping process models. A customer is in need of a software program but only knows which basic features they want to be included; finer details and details of functionality are not provided by the customer. The Waterfall model flows step to step through communication, planning, modeling, construction, and deployment. This structure is a very simple and straightforward layout that is usually best managed with smaller projects. The Prototyping model is commonly used for projects with undetailed requirements, quickly iterates through the design process (communication, planning, modeling, construction, deployment), repeating until the customer is happy.

To best help this customer, the software design process team should implement both of these methods. The Waterfall model requires sequential work which is often not possible to achieve. Therefore, the Waterfall model will be used as the structure and the Prototyping model will be implemented throughout to fully satisfy the customer. Starting with the communication step, the team meets with the customer and as detailed as possible, go through and discuss all the basic skeleton-software requirements for functionality throughout the software. After discussing the basic requirements, the team creates a detailed plan and a model. Rather than wasting time and resources on construction of a prototype, possibly going to be thrown away, the team brings their model to customer to discuss. If the customer wants to change something, they team plans and models again and meets with customer for approval. If the model is ok, the customer discusses any remaining features that they want include. The team repeats the process of planning and modeling until the customer is happy with all feature. Then the team moves onto the

construction of the prototype, and meeting with customer for input. If the requirements are all met, the team then finally moves onto the Deployment phase. Here, the combination takes the benefit of Prototyping’s iterative manner, ensuring communication throughout the process and ability for change; without wasting time and resources creating unwanted prototypes. It also keeps the overall structure of the Waterfall model, ensuring streamline efficiency.

(2.9) What are the advantages and disadvantages of developing software in which quality is “good enough”? That is, what happens when we emphasize development speed over product quality?

Developing software that’s quality is “good enough” means focusing on efficiency over quality of the software, meeting the bare minimum of requirements. Some people develop software in this manner in order to meet deadlines, or to cut costs for the customer. In addition to doing so in order to meet deadlines, it also allows for improvement. This is done to show growth of the software for stakeholders. However, this development process risks potential for bugs, faults and failures in the software. Meaning, developing software that is “good enough” for Mission Critical systems including autopilot, cruise control, and other life-dependent software should not be done.

Advantages	Disadvantages
- Quick delivery of software	- Risk of software failure
- Requirements all met	- Lower quality
- Costs cut (development & maintenance)	- Higher cost of maintenance – may need to re-develop multiple portions of code to fit update of requirement
- Allows room for improvement	- Not reliable

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

Agility in software projects is the ability to be “quick and ready.” An agile software development team would be able to quickly develop and complete requirements from customers while also being ready and able to adapt to any changes that may arise including new requirements from the customer or stakeholders. Although agile means quick work, that work must be complete and meet the simplest requirements. Simplicity is essential for agility in order to allow for changes and growth. Intricate code and development without a clear plan would causes issues for future adaptation.

Agility is also the use of smart and thought-out decision-making with communication constantly. Communication allows for free-flowing change rather than an abrupt stop in a more linear design process. A team can move rapidly to complete the software, but without organization and communication, the development process has a higher chance of getting off goal, wasting time and destroying the agile design process.

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

From my own personal observation of excellent software developers, they all posses three similar traits; detail-oriented, strong team member, and creativity. Detail-oriented developers are useful in ensuring that every requirement is met in full ability. Focus on the details also ensures a happy customer. A strong team member is necessary to maintain a productive and efficient environment for the team. While creativity in a developer brings new ideas and possibilities to every project. These three characteristics are all essential to excelling in software development.

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

There are eight core principles essential to guide process emphasizing the importance of efficiency, mobility, and creativity. However, of the eight core principles, there is one that stands out among the others and this is principle #4; build an effective team. While this may not look like the obvious choice for the most important principle, building an effective and strong team is what creates the possibility for a successful development process. Without strong team members, incomplete or inefficient work can create weak spots in your software leading to future problems. A strong team would also be comprised of people from varying backgrounds that work well together. Different backgrounds of people provide the ability to combine strong their attributes and set an example for the other members. A detail-oriented team member could help ensure principle #2 and #5; quality and establishing mechanisms for communication and coordination. A single team member could bring strong abilities in a certain area that can lead the way for other members, but a whole team of strong members would create a self-sufficient and nearly perfect development process.

(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?

Many software developers do not pay enough attention to requirements engineering because it is a very tedious and difficult task to complete and requirements can be unclear, missing, or constantly changing. Constant changes make for difficult planning for the requirements and many people would rather not do work for it to be tossed out the next day. They are however, a necessary step to ensure accurate software for the client. Without them, the software may not properly implement its different classes or forget to include a want of the client.

Developers are concerned with creating a working system. The only feasible scenarios for this process of development is if the developers are creating simple prototypes and not the intricate full software. Small prototypes made before the requirements are determined, is often used to test feasibility of the intended software. These small prototypes however, must be simple and quick in order to not delay the development process.

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

Use case: Withdrawal from ATM
Primary actor: The account holder
Goal in context: The account holder withdrawals money from the ATM
Precondition: The ATM is in good condition, and account holder has appropriate funds.
Trigger: Account holder needs $50 in cash from the ATM

1. The account holder inserts his debit card into the ATM.
2. The ATM’s card-reader sensors retrieve the account holder’s account information.
3. ATM gives visual feedback to the account holder informing them to enter their personal identification or pin number.
4. The account holder enters their pin into the ATM interface.
5. ATM displays compares entered pin to the account holder’s information for authentication.
6. ATM accepts pin number entered.
7. ATM displays possible options for the account holder to choose.
8. The account holder selects the “Withdrawal” action.
9. ATM displays message to account holder to input the desired amount the withdrawal.
10. Account holder inputs 50.00 on the ATM interface.
11. ATM checks account holder’s information to ensure they have enough money in account for the withdrawal.
12. ATM subtracts $50.00 from the holder’s account.
13. ATM dispenses $50.00.
14. ATM displays message to account holder of the withdrawal.
15. ATM dispenses account holder’s debit card from the reader.
16. Account holder takes money and card from ATM.
17. ATM clears the previous account holder’s information from its system.
18. ATM remains ready for next account holder.

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

During the software development process, it is possible to begin coding immediately after creating a requirements model. Doing so, however, sets up the process for disaster. A requirements model is not a complete model of a system. It contains on what the developers need to include, not which way or how they need to do it. Organization of the design is necessary for the process. It includes the smaller details like data types, what form we are collecting our data in and from which sensors is which data from. These important details are developed during the Design phase of the process which comes after the requirements phase. If a team were to begin coding with only their requirements model done, details would not be established, and goals would not be met. Coding needs to have a structure and logic to it or it will fail.

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

A sequence diagram shows the execution for one specific function(Use Case) and the objects(actors) that are involved in the execution of that function. Beginning with the initial interaction with the actor, the diagram traces the path of the use case. A state diagram is very similar to a sequence diagram, showing the function of a system with interaction from any actors. However, a state diagram shows *every* possible state and actor for a system, while the sequence diagram traces a singular state of the system. The state diagram is an overall view of the functions of the system, and the sequence focuses on one specific function. An example of a sequence diagram for a parking pay-station system could trace the sequence of an actor adding time to their parking spot. The state diagram for this system would should every possible state (add time/money, print ticket, pay) the system could enter from an actor.