CS-255 – Nicholas LaMantia – Quiz 8-3 – December 14, 2023

The next several paragraphs are referencing question one. First up is direct. With the direct approach, the old system is turned off and the new one is turned on. The advantages to this are that you no longer must worry about paying for the old system once it’s turned off. Instead, all you should worry about is if the new system works and is being paid for. The disadvantage of this approach is that if the new system does not work correctly, everything fails. As well, the new system might cost more than the old system whether that’s because of new technology, new encryption patterns, etc. It ultimately costs to implement, however this approach is less expensive than some of the others. In summary, the direct approach is very quick to implement and is low in cost, however it’s risky because the new system may be full of bugs.

Next comes parallel. This is where both systems are run until management decides that the old system can be turned off. This is very, very expensive and should be avoided. Its risks are that it’s very costly to the company; to run and manage two systems includes paying for the infrastructure and people to run it, and other costs which would otherwise be unnecessary. Though this may be the safest approach, it is very costly, and there is no definite end. It only stops when management decides to turn off the old system, which may take time, and therefore money. It is also complex, managing two systems at once.

After parallel comes single location. What the single location approach does is it tries out the system in a pilot project at one location, then implements it elsewhere if the pilot is successful. Notably, setting up a pilot project may be a lot of work which means you’ll have to pay your employees more (possibly). Nevertheless, it can lead to a safe installation to a system, safer than direct anyways. Though it is complex and may cost a sum, there isn’t much risk to the business.

Lastly, there is the phased approach, where the new system is brought online gradually, function by function in most cases. It would make sense to me that this is the safest route to development. Implementing one feature at a time makes it so that if there are bugs, the development team will know where to find the bug. It may be expensive to push features and functions one at a time over the course of months (for example), but there is very little risk involved. As well, there is little complexity, as the team is focusing on one aspect of the new system each time, whereas in other approaches, the team would be focusing on the whole new system, or worse, both systems at the same time. Agile software production does exactly this. It focuses on key features and develops them one at a time through sprints. One example of this is the implementation of a website. One sprint might focus on the UI of the website, while the other might focus on a database to store user information such as log in information.

The next paragraphs relate to the second question. The number of latent defects refers to the number of unknown errors existing in the system after it is installed. These errors are what make up most of the maintenance costs of maintaining a system. Now, SDLC is the architecture with which projects and systems are built around. The steps in order of SDLC are analysis, design, implementation, testing and integration, and maintenance. Analysis may be where errors can occur. For example, the customer might say that they want something in their product. This might come across as a need, or it might be overlooked. That possible requirement then escapes the analysis phase and makes its way into the design. This is where all the diagrams are laid out. If there is a latent defect within the system, it can probably be tracked either to bad code, or to one of the diagrams, which in turn is because of a bad requirement. If the defect isn’t recognized through the design process, then it makes its way into the implementation. It has a high chance of being recognized here as this is where we implement the system. An example of this may be related to the DriverPass system. If the customer didn’t specify that they wanted user data to be stored in a database, then there will be huge issues in terms of designing the system and the flow of data. If there are errors in the requirements, it will appear during deployment and sustainment. It is best to recognize these problems before they hit the next stage in SDLC so that costs can remain low. The less latent defects, the better. Developers can contain these defects by going back through the diagrams and requirements to see whether the data is flowing as it should, and whether the diagrams and requirements make sense. As well, making sure the code is well-written and works as it should helps to contain defects. Lastly, the cost of fixing an error increases exponentially based on the stage the error actually occurred from and when the error was discovered.

We can manage software volatility by setting expectations as to what systems we are using to develop our system, when those systems are updated, etc. As well, we can include testing of functional and non-functional requirements to test if the requirements are valid. This method can be used to identify defects so that the appropriate teams can correct them before they escape to later stages. The cost of latent defects which are discovered in the testing phase but were from the requirements phase (analysis) is great. It is very, very costly to fix errors which are discovered that late in the SDLC.