**1. State why it is a good idea to test a module in isolation from other modules and why different modules making up a software product are almost never integrated simultaneously?**

Within the software development life cycle, the design of software becomes realized and turned into source code during the coding and unit testing phase, or implementation phase (CSE ITT, Kharagpur, n.d.). Specifically, each of the software’s individual design components gets coded into separate modules. This is because it is more efficient to both code and test these modules in isolation. That is, it is easier to code isolated parts of software separately than it is to code a complex system all at once. More importantly, it is easier to test and debug any errors in the individual modules at this stage rather than wait until modules have been integrated into a larger, more complex system, where finding the culprit responsible for a software bug would be harder (CSE ITT, Kharagpur, n.d.).

At the end of the implementation phase, the result is a set of isolated, tested, and debugged program modules (CSE ITT, Kharagpur, n.d.). At this point, the modules can be integrated together into a larger system. However, this integration needs to be done in a planned and controlled way. Specifically, it is wise to integrated modules into the overall system incrementally, one piece at a time. As modules are added to the larger system, the system can be tested. In this way, any problems with the larger system can be isolated to the integration of the specific modules. Once all of the modules have been integrated into the larger system successfully, then a global system test can be conducted to ensure that SRS document specifications have been met by the working system (CSE ITT, Kharagpur, n.d.).

**2. Why are iterations usually limited when the waterfall model is used?**

By definition, the waterfall model does not make accommodations for requirement changes or iterations of the software. The waterfall model consists of separate and sequential phases, including the needs analysis and definition, the design stage, implementation and module testing, module integration and system testing, and, finally, the operation and maintenance stage (Sommerville, 2011). The waterfall model is rather inflexible because, in theory, each step of the model must be completed before the next phase can be moved to. Moreover, the waterfall model does not make plans for accommodating changes to the initial plan. In other words, the traditional waterfall model is meant to be followed from start to finish without moving backwards, or iterations.

Of course, no software development project can strictly follow the idealized, theoretical waterfall model (Sommerville, 2011). In practice, there are always changes to the initial development plan. These changes can be due to the changing needs of the customer or simply due to the changing technologies that support the development project, like new applications and system software. Regardless of the changes needed, the idealized waterfall model does not accommodate them. As such any requirement changes and consequential iterations when trying to follow the waterfall model will likely have a larger monetary cost than if the changes and iterations had been anticipated (Sommerville, 2011). While the waterfall method is useful in understanding the development stages, it should not be a shock that other models which accommodate changes and plan for iterations are more often used in practice.

**3. Why is it increasingly irrelevant to distinguish between software development and evolution?**

According to Sommerville, the distinction between software development and evolution is becoming increasingly irrelevant because “fewer and fewer systems are completely new” (2011). In order to evaluate this thoroughly, we first need to understand the terms development and evolution in the context of building software. Development is the process of building a software product and can encompass the assessment of needs, design, coding, testing, and maintenance. If we follow the traditional waterfall model of system development, these tasks are done sequentially and lead to the building of a new and completed working product. However, we have already determined through our studies this week that the traditional waterfall model is not practical. Instead, iterative and spiral models do a better job of ensuring that the customer’s needs are met by the end product. In this sense, development is no longer a tightly planned process that moves from concept to product realization. Instead, products are built in stages and changes are accommodated as customers’ needs and/or technological advances dictate. In fact, many software systems are simply built from the components of earlier products. Considering that evolution is defined by Sommerville (2011) as “changing the system in response to changing customer needs”, development is indeed approximating evolution as the methods to create software are continually adapting to changing technologies and customer needs.

**4. Explain why change is inevitable in complex systems and give examples (apart from prototyping and incremental delivery) of software process activities that help predict changes and make the software being developed more resilient to change.**

Changes are unavoidable and traditional software development models do not accommodate changes well (Aggarwal & Singh, 2005). Sometimes there are changing business requirements and sometimes underlying technologies change, forcing a change in the system development plan. The Spiral development model attempts to factor in so called “project risk” into the development process (Aggarwal & Singh, 2005). The idea at the heart of the spiral model is that the iterative development process should not only reassess customer needs periodically and accommodate them, but each iteration should also be evaluated for potential risks to the overall project. Moreover, during the risk analysis, not only are possible risks identified, but there is an attempt to mitigate them by finding alternatives and or resolving those risk factors. One benefit of doing risk analysis is that risks that can can threaten the cost of the project or the operation of the software can be identified and eliminated before they become problems (Aggarwal & Singh, 2005). Ofcourse, not every problem can be identified in advance, but having risk assessment early in project development and at every iteration of the system makes it more likely that potential problems will be found and headed off.

References

Aggarwal, K.K. & Singh, Y. (2005). Software Engineering: Programs, Documentation, Operating Procedures. Ed 2. Retrieved from <http://books.google.co.in/books?id=ytdKQGJ8f_AC&printsec=frontcover#v=onepage&q&f=false>

CSE ITT, Kharagpur (n.d.). Module 2: Software Life Cycle Model. Retrieved from

<http://nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Soft%20Engg/pdf/m02L03.pdf>

Sommerville, I. (2011) "Chapter 2 - Software Processes: Lecture 1 -2.” Software Engineering, Ed. 9. Retrieved from<http://my.uopeople.edu/mod/resource/view.php?id=57330>