HW4 - Foundations of Software Engineering Submitted by: Umang Garg

Q-16.1 Discuss the three "parts" of a design pattern and provide a concrete example of each from some field other than software.

A- The three parts of a design pattern are the context, the problem and the solution. Context is the environment in which the problem resides. The problem is when to apply the pattern within that context and the solution is a general arrangement of elements that helps in solving the problem.

An example of this from a field other than software could be making clothing patterns. One could make a clothing if the route of scissors through the piece of cloth is specified by providing the length of cuts and angles i.e. a pattern is provided and the type of cloth to be made is specified.

Design pattern identifies the key aspects of a commonly occurring design structure that makes it possible to create reusable OO design.

Q-16.4 What is framework and how does it differ from a pattern? What is an idiom and how does it differ from a pattern?

A- Frameworks are implementation-specific skeletal infrastructure that provides the generic structure and behavior for a family of software abstractions. Frameworks are written in programming languages. It is a large entity comprising of several design patterns. Framework are concerned with a specific application domain e.g. database, web application. Design patterns are more abstract than frameworks. Design patterns represent proven solutions to recurring problems that arise when developing software within a particular context. Although both design patterns and frameworks are mechanisms used to capture reusable designs, they are quite different. On one hand, design patterns are schematic descriptions of reusable designs that are not concrete programs and that are language independent. On the other hand, frameworks are compilable programs written in a specific programming language and often contain cooperating classes.

A programming idiom is a means of expressing a recurring construct in one or more programming languages. An idiom is something small as compared to a pattern. Idioms are more language specific as compared to patterns which are language-independent. A programming idiom is an expression of a simple task, data structure or algorithm that is not a built-in feature in the programming language being used or the use of an unusual or notable feature that is built into a programming language. An example of an idiom is expressing a way to increment a variable by one. i = i + 1 can be expressed as i++ idiomatically.

Q-20.1 Explain the difference between an error and a defect.

A- A defect is a quality issue in the software that is detected after the software has been released to the stakeholders or end users whereas an error is a quality problem in the software which is detected by the software engineers (or others) before the software is released to the end users. An error is usually a mistake by a programmer in coding. Defect is the absence of quality characteristic from its specified value which results in a product not satisfying its normal usage requirements.

Q-20.2 Why can't we just wait until testing to find and correct all software errors?

A- Testing from the early stages of construction helps in reducing the cost and time to re-work because the errors are not propagated to the next steps in the software process. Testing can be started from the Requirements phase and can be continued till the Deployment phase of software development. Testing also depends on the process model chosen. For example, formal testing is carried out in the testing phase in the waterfall model whereas in the incremental model, testing is performed at the end of every iteration phase and the whole application is tested at the end.

The verification and analysis of requirements are considered as a part of testing. Similarly, in the design phase, reviewing the design in order to improve the design is also categorized as testing. Testing conducted by the developer on completion of the code is also considered as testing.

It is a good practice to carry out testing simultaneously during every phase because it helps in the early discovery of errors and helps in reducing the overall cost of developing the software.

Q-20.9 A formal technical review is effective only if everyone has prepared in advance. How do you recognize a review participant who has not prepared? What do you do if you're the review leader?

A- Review participants have to prepare a review report/written notes of all the defects found and bring it to the meeting. If a reviewer hasn't prepared for the technical review, he wouldn't be familiar with the product and won't be able to contribute to the technical review.

It is the responsibility of the review leader to carry out the review meetings properly and to ensure that the review participants are prepared in advance. He should prepare a checklist of issues to be focused upon and if he identifies that a review participant has not prepared in advance, he has the right to question or inquire the participants. At the end of each review meeting a decision has to be taken by the attendees of the FTR (Formal Technical Review) on whether to accept the product without further modification or to reject the product due to severe errors or to accept the product provisionally. The decision taken at the end is signed off by all the FTR attendees.

Q-20.10 Considering all of the review guidelines presented in section 20.6.3, which do you think is most important and why?

A- The most important guidelines for a successful FTR are:

- Set an agenda and maintain it and
- Limit the number of participants and insist upon advance preparation

An FTR should be conducted properly, errors should be pointed out and the tone of all the FTR attendees should be constructive and soft. FTR must be kept on schedule. It should be the responsibility of the review leader to maintain the meeting schedule. He should ensure that the meeting is on track and not digressing away from the main issues.

It should be essential that the review participants are prepared well in advance. If advanced preparation is not done, a review participant won't be able to contribute to the FTR meeting. Also, the number of participants should not be too less and should not be too many.

Q-18.1 Explain why deciding to develop MobileApp for several devices can be a costly design decision. Is there a way to mitigate the risks of supporting the wrong platform?

A- Mobile devices come in a variety of screen sizes as compared to personal computers which requires greater attention to UI design issues. Intermittent connectivity outages, limited battery life should also be taken into account while designing MobileApps. Software developers must design algorithms which are energy efficient in order to conserve battery life. Proper design trade-offs between the expressive power of MobileApps and stakeholder security concerns should be identified. Portability should also be an important consideration while designing a MobileApp because they have to execute on a number of device platforms. Accommodating all these factors on multiple platforms require additional time and efforts which results in an increase in the overall project cost.

A way to mitigate the risks of supporting the wrong platform is by prioritizing the factors that are required according to the market demand. These factors could be Speed/Performance, Cost, Standards/APIs, User experience, etc. With the understanding of the priorities, the factors can then be compared to the platforms that are available and the platforms that work more efficiently with the factors with the highest priorities should be used for development first.

Q-18.3 Add at least five additional questions to the MobileApp Design – Quality Checklist presented in Section 18.2.

A- The five questions that could be added are:

1) Are the design algorithms for the software energy efficient to conserve the battery life of the device?

2) Does the MobileApp has support for multiple languages?

3) Does the MobileApp deliver personalized user experience based on the physical location of the mobile device?

4) Are the interaction possibilities (e.g., voice, touch, gesture, eye tracking) associated with a mobile device been used appropriately?

5) Have the developers used programming shortcuts to reduce the demands made on processor and memory resources?