Jasir Massey Campbell

Software Engineering

Assignment 5

**9.2.** Present two or three examples of applications for each of the architectural styles noted in Section 9.3.1.

Data-centered - grade-book, hospital patient database,

Data-flow - calculator, random-name generator, graphing software

Call and return - query, function modules/methods, client-server

Object - oriented - java, c++

Layered - software that uses the techniques encapsulation coding, inheritance coding

**9.3.** Some of the architectural styles noted in Section 9.3.1 are hierarchical in nature and others are not. Make a list of each type. How would the architectural styles that are not hierarchical be implemented?

| hierarchical | not hierarchical |
| --- | --- |
|  | data centered |
| object oriented | call and return |
| layered | data flow |
|  |  |

Although data-centered is not necessarily hierarchal, it can be in a certain respect. If privileges determined how much more or less access a user had to that database compared to another, it can be argued that hierarchy exist within a data-centered system.

**9.8.** Using a data flow diagram and a processing narrative, describe a computer-based system that has distinct transform flow characteristics. Define flow boundaries and map the DFD into a software architecture using the technique described in Section 9.6.1.

**10.1.** The term *component* is sometimes a difficult one to define. First provide a generic definition, and then provide more explicit definitions for object-oriented and traditional software. Finally, pick three programming languages with which you are familiar and illustrate how each defines a component.

Components are aspects of software that can be labeled and that serve a specific purpose within the software. For example, components of software could be a database or method. A database could be an important component when explaining the general composition of a software to a stakeholder that is less technically literate. But for the developer this information is not enough. The developer will want to know what are the components that compose the database? What functions/methods are needed to manipulate the data in the database? The way that engineers often look at software development is in layers. If a developer was making a program called DOG and wrote a description for it. It may look like this.

class DOG {} - classes are great in assisting programers with avoiding redundancy, encapsulation and controlled data abstraction through the uses of public and private members.

Class DOG might contain some properties to describe every dog such as:

const int legs = 4;

void bark();

Obviously not all dogs are the same, so another component that can be added to the program is inheritance. With inheritance we can create smaller more specific classes that derive and adopt all of the properties of DOG{};

class PitBull::DOG{};

Within this subclass we can specify properties that a specific to pits. To conclude, some of the different components in our program include: classes, subclasses, static properties, methods. All of which can be defined in different ways to provide meaning to different stakeholders.

**10.2.** Why are control components necessary in traditional software and generally not required in object-oriented software?

In basic terms, the reasons rests within the main usage of the two software. With object oriented, the functionality is implicitly defined for each component whereas with traditional software, it is not.

**10.5.** Select three components that you have developed recently and assess the types of cohesion that each exhibits. If you had to define the primary benefit of high cohesion, what would it be?

Calculator App

1. User Input
2. Programmatic Computations (add, subtract, divide, multiply)
3. UI Display Output

This is an example of functional cohesion. A computation is performed and result returned. High cohesion basically refers to all functions and members having purpose. As long as all the functions and attributes of a class is associated with the class, high cohesion exists.

**10.6.** Select three components that you have developed recently and assess the types of coupling that each exhibits. If you had to define the primary benefit of low coupling, what would it be?

Low coupling has a bunch of advantages including such as making it easier for programmers to follow to flow of data during debugging or editing, or it can increase class independence and reusability. Module independence is important because it makes it easier for the software engineers to design, write and test code without having to worry about if the removal of one module will effect the rest of the code.

**11.2.** Develop two additional design principles that “place the user in control.”

1. Logout
2. Delete Account

**11.3.** Develop two additional design principles that “reduce the user’s memory load.”

1. Delete
2. Server-Side Functionality

**11.4.** Develop two additional design principles that “make the interface consistent.”

1. Color Scheme / Use of UI Kit
2. Use of UI Diagram

**11.7.** Add at least five additional questions to the list developed for content analysis in Section 11.3.3.

1. How does the data look when unpacked on different devices?
2. Will videos be allowed to be screened in different formats?
3. Are globally recognizable icons used to promote simplicity for the user? (ex. cog icon for settings)
4. Will colors be used to promote a specific feeling, emotion or setting?
5. Will pictures and video auto reformat to display the best way to user?

**11.12.** Provide a few examples that illustrate why response time variability can be an issue.

Variability refers to the time that represents the difference between present system response time and the average system response time. The biggest issue with variability is that humans, even when the few who may understand why their response times are choppy, are extremely impatient. And as bad as the user may wish they had a lower response time than their highest variability, consistency is also key in keeping a happy user.

**11.13.** Develop an approach that would automatically integrate error messages and a user help facility. That is, the system would automatically recognize the error type and provide a help window with suggestions for correcting it. Perform a reasonably complete software design that considers appropriate data structures and algorithms.

**11.14.** Develop an interface evaluation questionnaire that contains 20 generic questions that would apply to most interfaces. Have 10 classmates complete the questionnaire for an interactive system that you all use. Summarize the results and report them to your class.

1. Is the product easy for the user to learn?
2. Are object on interface efficiently spaced and played out?
3. Are colors used ti invoke specific feelings?
4. How often will users encounter errors?
5. What type of errors are likely to be encountered?
6. Is the product easy to use?
7. Does the interface use other device such as a camera that needs screen involvement?
8. How can consistency be evaluated?
9. Does the interface offer multiple ways to accomplish tasks?
10. Is there help documentation?
11. Is it aesthetically pleasing?
12. How do errors interrupt ongoing tasks?
13. Is minimalism a focus?
14. How is user freedom measured?
15. Can users report problems?
16. Does it auto format on different device?
17. Does it have multiple device capability?
18. Is there a search function?
19. Is there a menu bar?
20. If so, is it dynamic?

**12.2.** What is the difference between a non-generative and a generative pattern?

Both generative and non-generative patterns describe reoccurring phenomena that can be found throughout the software engineering process. The difference between the two is that non-generative patterns do not specify the solution to actually create it.

**12.3.** How do architectural patterns differ from component patterns?

The difference between architectural patterns and component patterns is in the scope in which they cover. Software architecture patterns describe the entire scope of the software whereas component patterns describe just a single “component” of the software. In simpler terms, architecture patterns describe reoccurring solutions for larger software problems, and components describe solutions for smaller software problems.

**12.4.** What is a framework and how does it differ from a pattern? What is an idiom and how does it differ from a pattern?

Both focus on promoting reusability but the biggest difference between a framework and a pattern is their scope. Frameworks often have a much larger scope than patterns. They include multiple sets a classes and libraries, and function as working templates that can be reused and modified to create a different/modified software. Patterns are documents that describe solutions to reoccurring design problems. Unlike frameworks which are designed to be used in multiple projects, patterns are developed and used for specific projects. An idiom is

**12.7.** Find five patterns repositories and present an abbreviated description of the types of patterns contained

DBContext

Entity

IQueyrable

Commands and Queries

**12.12.** The gang of four [Gam95] have proposed a variety of component patterns that are applicable to object-oriented systems. Select one (these are available on the Web) and discuss it.

Factory Method Pattern - According to refactoring.guru, “**Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.”

**12.13.** Find three patterns repositories for user interface patterns. Select one pattern from each and present an abbreviated description of it.

Value Models - <http://c2.com/ppr/vmodels.html#what_is_a_valuemodel>

**12.14.** Find three patterns repositories for WebApp patterns. Select one pattern from each and present an abbreviated description of it.

Login Pattern - <http://www.welie.com/patterns/showPattern.php?patternID=login>

Comment Box - <http://www.welie.com/patterns/showPattern.php?patternID=comment-box>

Testimonials - <http://www.welie.com/patterns/showPattern.php?patternID=testimonials>