**CS 4321 - Video Lecture Expectations**

**Video P3L1: Architecture (40 minutes)**

Watch video P3L1 on Udacity. Answer the questions below and submit on Blazeview (HW-P3L1). Instructions:

* Do not remove the questions.
* You can provide the answer(s) where the blank is, but preserve the underline (or use a different color for the answers)
* Or, you can provide the answers below the questions. For example, you could type: Answer: x, y, z.

**Questions to be answered**

Watch clips 1-20. Clips 21-23 are very interesting, but I won’t test from them.

1. When building a software system, you go through a process of making many design decisions many of which don’t affect the success of your system. Some, however, are the equivalent of a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in a building.
2. When we are doing maintenance of a system, often we make locally optimal decisions for any one of these changes with no over-arching vision of how we should do this and what it might affect. What you might end up finding is that the globally optimal behavior of the system is badly compromised. This is referred to as \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
3. For software engineers, architectural erosion often results in a huge \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ problem.
4. The author defines software architecture as the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ for the software system.
5. A software architecture (SWA) is not defined at once, but iteratively over time. At any point in time there is a SWA, but it will \_\_\_\_\_\_\_\_\_\_.
6. A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architecture captures the design decisions made prior to the system’s construction.
7. A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architecture describes how the system has actually been built.
8. As a system evolves, the ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architecture should be modified first, but often isn’t.
9. The prescriptive and descriptive SWA’s tend to diverge in practice because of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
10. Architectural \_\_\_\_\_\_\_\_\_\_\_ is the introduction of design decisions that are orthogonal to a system’s prescriptive SWA and results in unnecessarily complex architectures.
11. Architectural \_\_\_\_\_\_\_\_\_\_\_ is the introduction of design decisions that violate a system’s prescriptive SWA.
12. When faced with a degraded SWA you can \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ or undertake \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ where you determine the SWA from implementation and fix it.
13. When two modules depend on each other, this is referred to as a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Read this before Clip 12 – Definition of *cohesion* and *coupling*

A method, class, package, component has *high cohesion* if it keeps together things that are related to each other, and keeps out other things. A module that has high cohesion has desirable traits such as: robustness, reliability, reusability, and understandability. For example, a domain class that encapsulates an Employee that also has methods to read data from a database would have low cohesion.

Coupling occurs when there are interdependencies between one module and another. It is a measure of the degree to which each module relies on the other modules. When interdependencies exist, changes in one place will potentially require changes somewhere else. Modules almost always have dependencies on other modules, however, we want to minimize these dependencies through the design process.

We want software modules to have high cohesion and low coupling.

1. Three ideal characteristics of a SWA are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
2. Two benefits of systems with low coupling and high cohesion are that they are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. A SWA is composed of 3 elements: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ elements, which implement the business logic; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ elements which contain the data; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ elements, which are the glue that hold the different elements together.
4. A software \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is an architectural entity that encapsulates a subset of the system’s functionality and restricts access to that subset via an explicitly defined interface.
5. A software \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is an architectural entity effecting and regulating interaction.
6. A software \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is an association between components and connectors of a SWA.
7. A UML \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ diagram is sometimes used to represent a software configuration.
8. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architectural style is one where the processing elements are arranged so that the output of one element is the input to another.
9. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architectural style is where some elements send messages, but not directly to specific receivers. Instead, they publish messages with one or more associated tags without knowledge of who will receive such messages. Similarly, subscribers will express interest in one or more tags and will all receive messages of interest according to such tags.
10. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architectural style is where some elements provide services and other elements initiate contact and request those services.
11. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architectural style is where individual elements both provide services and request services of other elements.
12. Skype is primarily utilizes a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architecture with some minimal elements of a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ architecture, for instance to start the service you login to a server.

Comment: Architecture in practice

The architecture of most systems has aspects of several different styles. For example a Gui system (event-driven architecture) is many times also a client-server system. In addition, it could also contain components that are pipe-and-filter.