**WHAT IS ARCHITECTURE?**

A software architect is a programmer. The goal of the architect is to create a shape for the system that recognizes policy as the most essential element of the system while making the details irrelevant to that policy.

The architecture of a software system is the shape given to that system. The purpose of that shape is to facilitate the development, deployment, operation, and maintenance of the software system contained within it. The ultimate goal is to minimize the lifetime cost of the system and to maximize programmer productivity.

**Development**

**Deployment**

**Operation**

The impact of architecture on system operation tends to be less dramatic than the impact of architecture on development, deployment, and maintenance. Almost any operational difficulty can be resolved by throwing more hardware at the system without drastically impacting the software architecture.

Indeed, we have seen this happen over and over again. Software systems that have inefficient architectures can often be made to work effectively simply by adding more storage and more servers. The fact that hardware is cheap and people are expensive means that architectures that impede operation are not as costly as architectures that impede development, deployment, and maintenance.

A good software architecture communicates the operational needs of the system.

**Maintenance**

Of all the aspects of a software system, maintenance is the most costly. The never-ending parade of new features and the inevitable trail of defects and corrections consume vast amounts of human resources Of all the aspects of a software system, maintenance is the most costly. The never-ending parade of new features and the inevitable trail of defects and corrections consume vast amounts of human resources.

**Keeping options open**

Software has two types of value: the value of its behavior and the value of its structure. The second of these is the greater of the two because it is this value that makes software soft.

The way you keep software soft is to leave as many options open as possible, for as long as possible. What are the options that we need to leave open? They are the details that don’t matter.

All software systems can be decomposed into two major elements: **policy** and **details**.

The **policy** element embodies all the **business rules** and **procedures**. The policy is where the true value of the system lives.

The **details** are those things that are necessary to enable humans, other systems, and programmers to communicate with the policy, but that do not impact the behavior of the policy at all. They include IO devices, databases, web systems, servers, frameworks, communication protocols, and so forth.

The goal of the architect is to create a shape for the system that recognizes policy as the most essential element of the system while making the details *irrelevant* to that policy.

If you can develop the high-level policy without committing to the details that surround it, you can delay and defer decisions about those details (web server, type of database, dependency injection, etc.) for a long time.

What if the decisions have already been made by someone else? What if your company has made a commitment to a certain database, or a certain web server, or a certain framework? A good architect pretends that the decision has not been made, and shapes the system such that those decisions can still be deferred or changed for as long as possible.

A good architect maximizes the number of decisions not made.

**Device Independence:**

Card reader, junk mail, physical addressing (aghast) examples.

The open-closed principle.

**Conclusion:**  
Good architects carefully separate details from policy, and then decouple the **policy** from the **details** so thoroughly that the policy has no knowledge of the details and does not depend on the details in any way.

**Quiz:**

**1 The ultimate goal of a software architect is to...**

minimize the lifetime cost of the system and maximize programmer productivity.

support the life cycle of the system.

make the system easy to understand.

facilitate the development, deployment, operation, and maintenance of the software system.

**2 Use cases expect \_\_\_\_\_\_ and output \_\_\_\_\_\_.**

Entities; Entities

Request Models; Response Models

**3 Request and Response models should contain references to Entities.**

True

False