Table of Contents

"The Principles of Clean Architecture" by Uncle Bob Martin

Reference

The Principles of Clean Architecture by Uncle Bob Martin.

Notes

Good general advice on base architecture of an application.

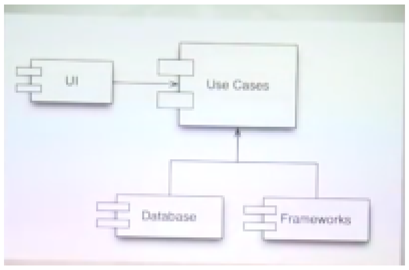
Message is that the architecture of an application should “scream” its intent. It should say “I am an accounting application” or “I am an order entry application”. It should not say “I am a data base application” or “I am a Ruby on Rails” application. The problem is, the main organization method that is presented to the developer is often “the framework” which does not tell you what the application does.

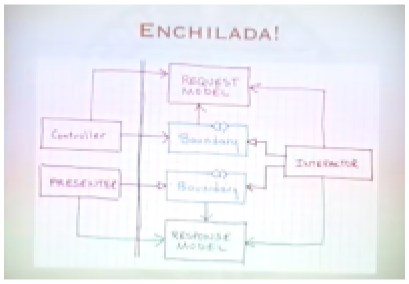
My view is that “architecture” is about making the process of delivering on customer expectations easier, cheaper, faster not only for the current release, but future releases as well. To me this means that good architecture is about setting the system up so that it can be changed easily.

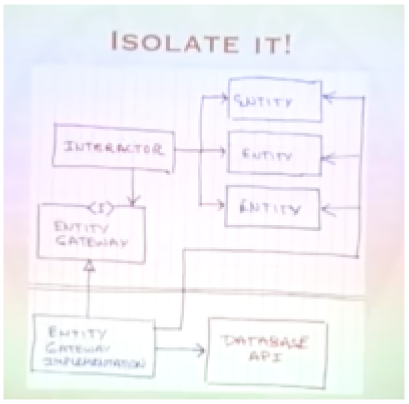
Uncle Bob provides a mechanism for doing this. His message is that “a good architecture allows major decisions to be deferred” (even decisions about what, if any, database to use or what if any framework to use) and that “a good architecture maximizes the decisions not made”. This means you should be using a “plugin architecture” so there are clean boundaries for things like “persistence” (which to many means “database” or especially “SQL database” but in reality is just putting stuff away in storage) and “user I/O” (which to many means “Web page” etc but really this is just I/O, but also applies to “SOA interaction, etc) between the business rules that define the application and the various levels of supporting infrastructure. One huge benefit of doing this is that you end up with an application which you can test automatically, quickly and repeatably since you can test all the plug in separately, which not only supports refactoring, but also supports change.

The other interesting point made by Uncle Bob is the use of SQL databases. His contention is that databases were put in place to help people deal with the complex problem of persistence when all we had was disk drives that needed complex mechanisms to address / find information and were relatively slow. Today we have solid state drives. The question he asks is “if your persistence model is a 2 ^ 64 addressable linear space that you can read / write to / from, do you need a relational database?” After all, the first thing you do when you read a database record is turn it into a hash table or linked list or something else that is more useful. Why not just store things that way? Why not just treat it like you would RAM? Of course even if we believed today that this thinking has validity, we'd need to also address the inertia of industry that has been built up around databases - including DBA's, software products, corporate perception of their 'data asset' - but it is worthwhile thinking about this and also worthwhile worrying about the related architectural elements. For example what does it mean to create "transactions" when you can just write to something you treat like RAM? If you are unsure, perhaps we should be isolating the database from the application … Good discussion.

Pictorially the recommendations looks like:







Video, Webinar, Learning, Architecture, Review