Modern Software Design

Who Am I ?

Ahmed Adel Ismail

Lead Android Developer

@Vodafone Shared Services Egypt

Highlights

2012 - Oracle Certified Professional Java 6 Programmer

In 2012 I passed the OCP Java 6 Programmer Certificate with score 98%

2015 - Best Application In Egypt at CBC Channel’s poll

I was responsible for “Bey2ollak” Application, which was selected as the best Application in Egypt at CBC Channel’s 2015 poll

2017 & 2018 - Ranked as one of the top 10 Java developers in Egypt

Ranked as one of the top ten Java developers in Egypt on Github as per Git-Awards website :

http://git-awards.com/users?country=egypt&language=java

What is Software Design?

Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints...

"***wikipedia.org***"

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation...

"***tutorialspoint.com***"

"Software Design is the skill of technical-decision making"

What Technical Decisions?

What

What will the software look like to achieve the requirements, this is called **Architecture**

How

How will the software achieve these requirements, this is called **Design**

Why

Why will the software choose a certain solution to achieve the requirements over another solution, this is called **Trade-offs**

How to design our Software?

Before we can be able to decide **what** will our software look like, we should be able to know **how** we can design software, minimum knowledge of some principles, and techniques is needed

Software Design Principles

YAGNI - You Aren't Gonna Need It

This principle was developed by Extreme Programming, which demands to never put any piece of code that is not needed now, even when we are designing our class, never put a non - private method that nobody will use it.

KISS - Keep It Simple and Stupid

Whenever we write a piece of code, it should be very simple and stupid, it should not hold complex logic or hard to read code, if we have a function that makes a complex algorithm, divide it into smaller functions, each function does a small and simple part of the algorithm, and each one of them is named clearly.

DRY - Don't Repeat Yourself

Try to avoid repeating code, never copy and paste code snippets, one of the key solutions to such problem in object oriented programming is design patterns, in functional programming, this is solved by encapsulating the code snippet inside a function.

DRY

LoD - The Law of Demeter

Also known as “principle of least knowledge”, this law says that a component or an object should not know about internal details of other components or objects, and this means that whenever we find ourselves using the *getters* of another Object to make some calculation, these calculations should be done by this other Object (or a function outside)

Single Responsibility Principle

Every part in the application should do one and only one thing, starting from the level of methods and functions, to classes, even to packages … everything should have only one reason to change

Open Closed Principle

Every class should be open for extension / inheritance, but closed for modification, which means that if we have a class that makes something, and we want to make this class do another thing, we should not modify this class, but we can create a subclass that does this new thing, or we can make extension functions for that class

Liskov Substitution Principle

When we use a subclass instead of the superclass, this behavior should not break the software, in other words, we do not override methods in our subclass and make it crash or misbehave when used

SOLID

Interface Segregation Principle

Never declare methods in an interface that one of it's implementing classes won’t need to override, same goes for abstract classes

SOLID

Dependency Inversion Principle

Try to deal with the parent types as much as possible, try to deal with interfaces or abstract classes as much as possible, this will make the software more flexible and easy to change

Software Design Techniques

Divide and Conquer

Divide the software into small, high cohesive parts, this helps working in parallel, also small chunks of code are easier to reason about, reuse, refactor, test, and follow the Single responsibility principle

Cohesion

Increase cohesion where possible, strong forms are :

- **Functional** (divide the software into functions)
- **Layered** (divide the software into layers)
- **Communication** (code using same data is kept together)
- **Sequential** (group by methods using results of previous one as the parameter of the next)
- **Procedural** (group by method invocation order)
- **Temporal** (group by method invocation timing)

Coupling

Avoid tight coupling where possible, tight forms are :

- **Content** (modifying inner data of another class)
- **Common** (global variables)
- **Control** (if-else blocks for method parameters)
- **Stamp** (method parameters as other types)
- **Data** (method has too many parameters)
- **Routine Call** (method invoking another method)
- **Type Use** (class declares another type as variable)

Abstraction

Keep the level of abstraction as high as possible, make sure that the code makes it easy to hide as much details as possible

Reusability

Increase reusability where possible, design the code so that it can be reused in multiple contexts, following the previous principles and techniques will help achieving this easily

Reuse existing

Reuse existing design and code where possible, reusing the existing code and design benefits from the investments of the others, but put in mind that to *clone* or *copy/paste* code snippets is not considered reusability

Flexibility

Increase the flexibility of the software to be prepared for future changes, which can be achieved by :

- Reduce coupling and increase cohesion
- Work with abstractions
- Never hard code anything
- Leave all options open, do not put limitations that hinders modifying the software later

Anticipate Deprecation

Prepare for changes, the more we use external code, the more often we will get hit with deprecated parts

- Avoid using early releases
- Avoid using undocumented libraries
- Avoid using software from companies that will not provide long-term support
- Use standard languages and technologies that are supported by multiple vendors

Testability

Design the code to be tested by another code, the more we divide our code, and deal with abstracts, the more easily we can unit-test it… one key for testability is dependency injection (no need for frameworks to implement this pattern), another key is pure functions

Design Defensively

Never trust how others will try to use our code, design the code in a way that guarantees that no one will use it the wrong way

What should our software look like?

Since we know **how** we can build a software, now we can think about **what** our software should look like … and this requires the knowledge of some "patterns", which are called *Architecture Design Patterns*, or *Architecture Patterns*

Popular Mobile Architectures

Clean Architecture

This Architecture is concerned about dividing the whole application into layers:

- Presentation
- Domain
- Entities

with respect to the dependencies direction, and dependency inversion

MVC

This Architecture is concerned with dividing the *presentation* layer into:

- Views
- Controllers

And a *Model* class that is the entry point for the domain layer

MVP

This Architecture is concerned with dividing the *presentation* layer into:

- Views
- Presenters

And a *Model* class that is the entry point for the domain layer

Popular Mobile Architectures

MVVM

This Architecture is concerned with dividing the *presentation* layer into:

- Views
- ViewModels

Where the ViewModel class is responsible to communicate with the domain layer

Why this Architecture ?

Now we know **what** are the possible architectures, and **how** can we achieve them … but **why** should we choose over another ?

** This branch of knowledge is not covered in these slides*

“Architecture does not care about how it can be implemented, it only cares about the pattern which the software will follow to achieve the business requirements”

“Design is the knowledge of achieving the desired architecture in a way that guarantees the best practices, which respects scalability, maintainability and other quality aspects”

Thanks!

Any questions?

Contact me

https://www.linkedin.com/in/ahmed-adel-183624a3/

https://github.com/Ahmed-Adel-Ismail

ahm3d.ad3l@gmail.com

+2 01000 3 7000 2