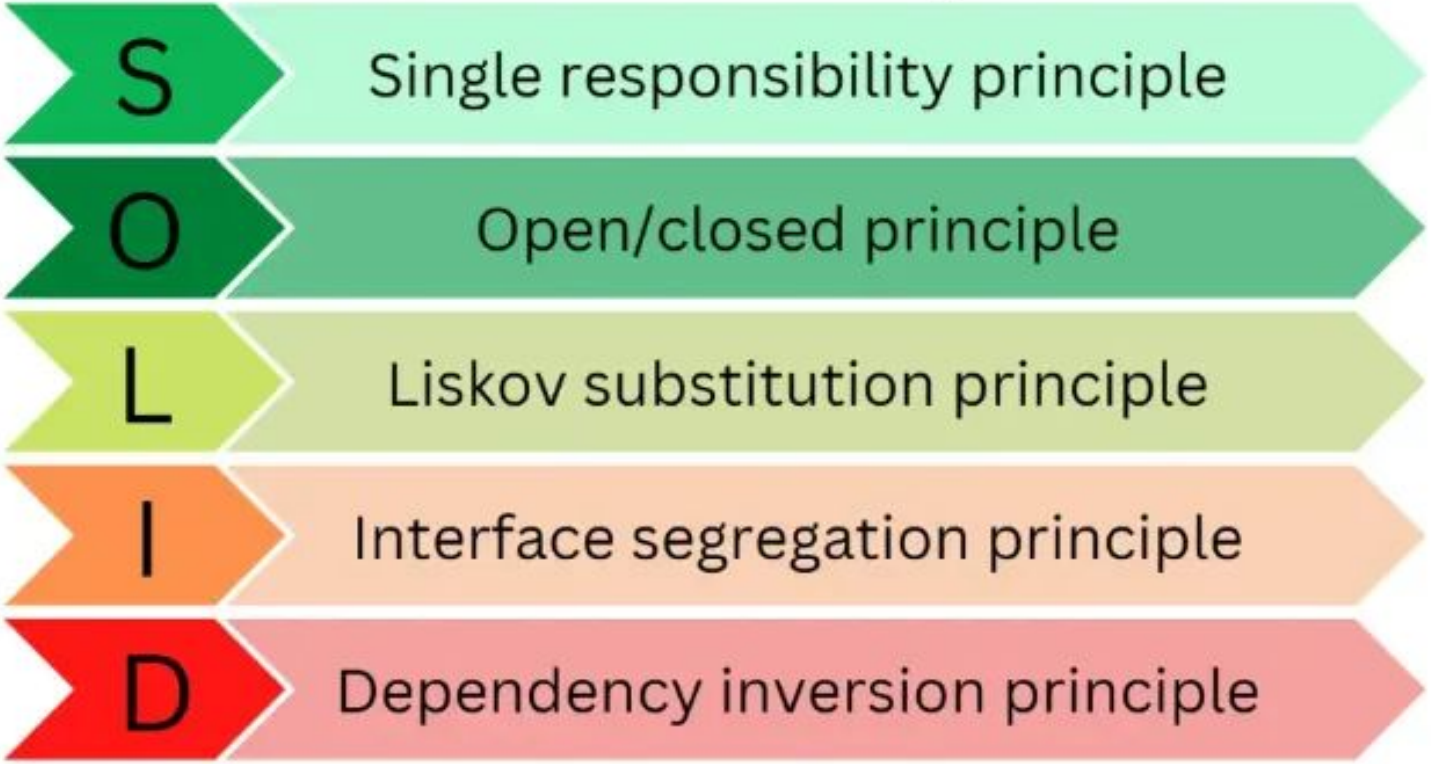
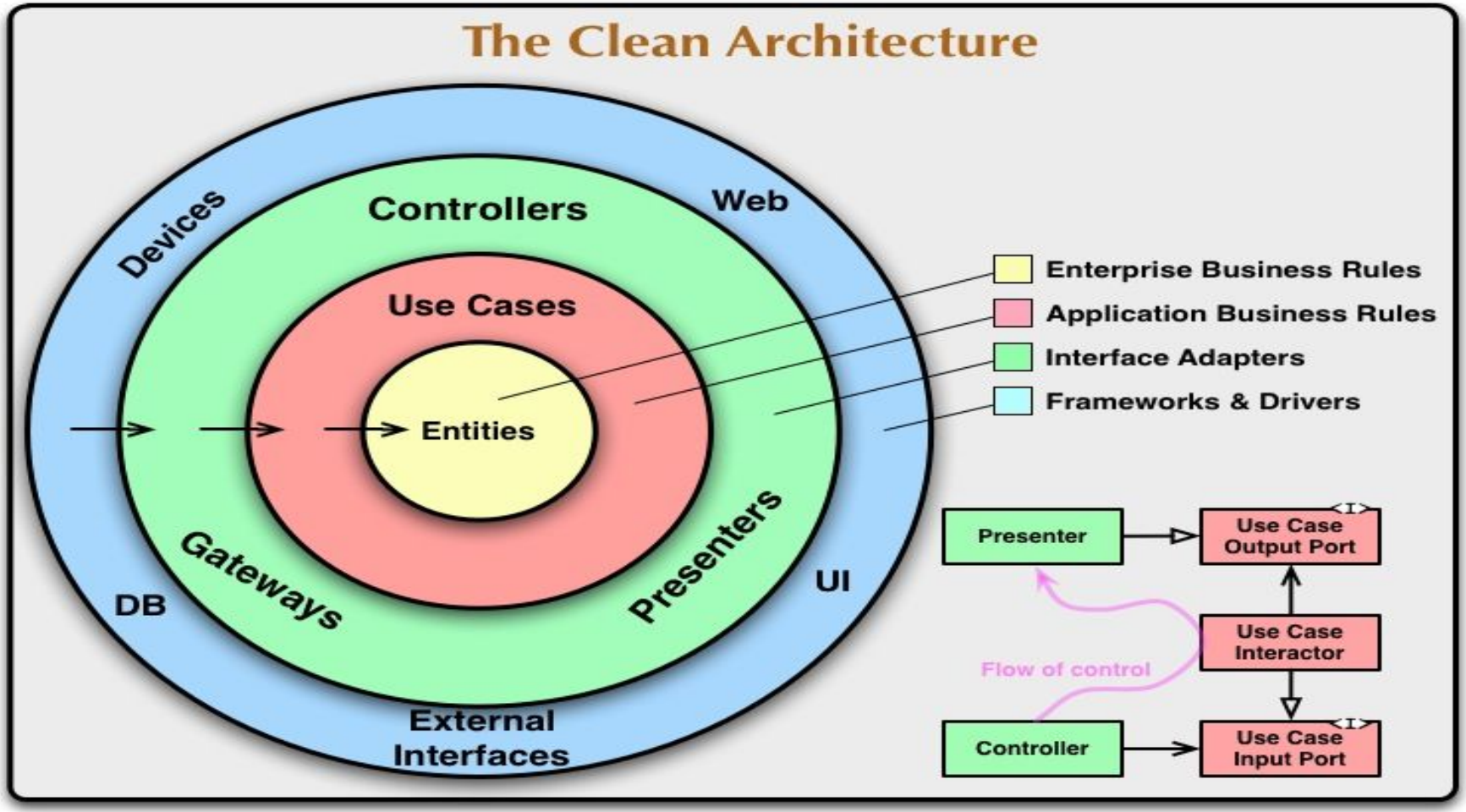
Test Automation Architecture

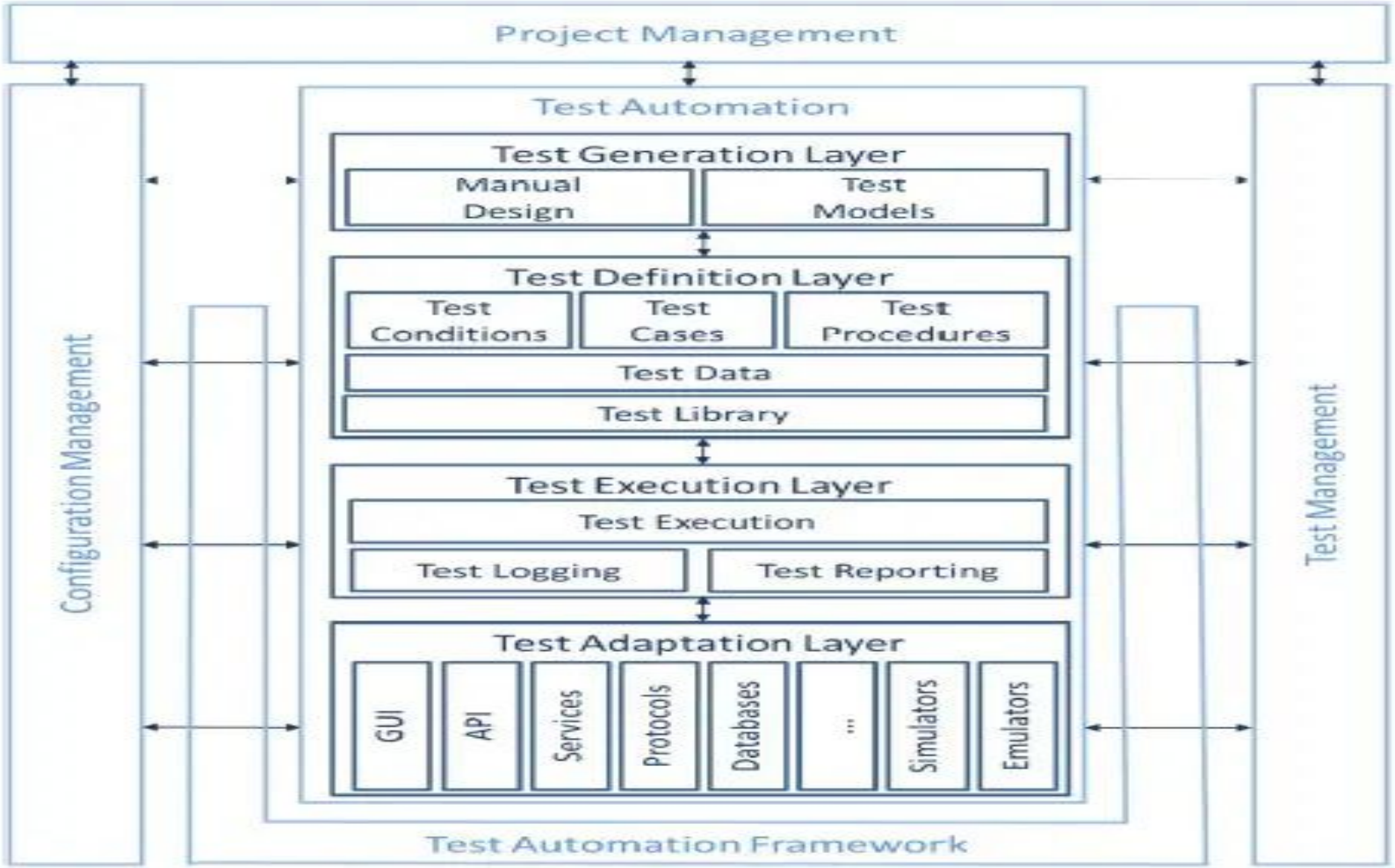
Challenges associated with Large UI automation projects

- Due to the size of a project and the number of test cases, test code becomes increasingly difficult to maintain and keep up to date to changes in the application. This is a time consuming task when the idea is to reduce time by automating your Regression suite.
- There is potentially a lot of repetition on a typical functional test suite, there are steps that are done once and again by different tests with little variance, some of this can be aided by using Data-Driven testing but that still leaves you with many different pieces of your code that perform actions on the same elements.
- If a change happens in the application, then all UI Element locators must be updated, increasing the risk of human error and maintenance effort.
- If there is no reutilization of code, new test cases take about the same time to develop than earlier test cases.

SOLID Principles




The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port


Project Management
Test Automation
Test Generation Layer
Manual Design
Test Models
Test Definition Layer
Test Conditions
Test Cases
Test Procedures
Test Data
Test Library
Test Execution Layer
Test Execution
Test Logging
Test Reporting
Test Adaptation Layer
GUI
API
Services
Protocols
Databases
...
Simulators
Emulators
Configuration Management
Test Management
Test Automation Framework

Design Patterns

- **Page Object Model.** The basic idea is to divide the application into modules/pages/panels as needed and abstracts object recognition and actions on those objects from the test level. I'll expand on that a bit later. POM is normally paired with other common elements such as Object Repository and Driver factory. If implemented efficiently, it really improves on the maintainability and reusability of code. It's the most used design pattern for UI automation, especially with Selenium-based frameworks.
- **ScreenPlay Model.** This model takes POM further by organizing the page objects, their actions and other elements such as inputs, goals, actor, etc. into a more readable (and supposedly maintainable) screenplay organization.
- **Façade Design Pattern**. It's similar to the page object model, but it's geared to full facades or forms in which many inputs and possibly more than one actions need to happen, the main drawback is that variance in the workflow force you to create another façade class. The test level calls the whole façade class and provides an object which contains all the inputs needed. This is similar to how API testing is sometimes organized.
- **Fluent Design Pattern.** It's a different flavor of POM that is supposed to conform better with Behavior Driven Development since it forces the test to be done in a "logical chain" or workflow. The page objects are written in an fluent interface manner in which methods can be cascaded or chained in a flow of calls. This is achieved by making the methods return an page class object of the type required to continue the flow. Example of how a POM test method would look like if using a Fluent design: