

Key concepts I

- ports and adapters
- hexagonal architecture
- outside-in thinking
- primary port
- secondary port
- adapter
- test double (mocks aren't stubs)
- three layers architecture

Key concepts II

- polymorphism
- modules: horizontals and verticals
- vertical slicing
- TDD vs Test first
- dependency inversion principle
- branch by abstraction
- technical debt
- parallel change
- feature toggle
- walking skeleton / service template

Ports and adapters

Ports and adapters

The main idea of Ports & Adapters is to define the structure of an application so that it could be run by different kinds of clients (humans, tests cases, other applications,â€™), and it could be tested in isolation from external devices of the real world that the application depends on (databases, servers, other applications...)

Hexagonal architecture

Hexagonal architecture

The same as ports and adapters.

Outside-in thinking

Outside-in thinking

Many developers focus on implementing the Domain Model before defining how it is going to be used by the external world. Outside-in thinking it's the opposite.

Primary port

Primary port

Primary ports are the main API of the application. They are called by the primary adapters that form the user side of the application. Examples of primary ports are functions that allow you to change objects, attributes, and relations in the core logic.

Secondary port

Secondary port

Secondary ports are the interfaces for the secondary adapters.

They are called by the core logic.

Example: an interface to store single objects.

Adapter

Adapter

An adapter is a bridge between the application and the service that is needed by the application.

It fits a specific port.

Test double

Test double

Test Double is a generic term for any case where you replace a production object for testing purposes.

There are various kinds of doubles: Mocks, Stubs, Fakes...

Three layers architecture

Three layers architecture

A three-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms.

Polymorphism

Polymorphism

The provision of a single interface to entities of different types.

Modules: horizontals and verticals

Modules: horizontals and verticals

Application layers: Presentation, Business, Data, etc...

Horizontal slices align with application layers, dividing workload, initiatives, and development resources amongst the individual slices.

Vertical slices divide the application layers vertically, the slices include all functionality of a particular feature from the back-end to the front-end.

Vertical slicing

Vertical slicing

A vertical slice is a portion of a game which acts as a proof of concept for stakeholders before they agree to fund the rest.

TDD vs Test first

TDD vs Test first

TDD is a design technique. You write the tests before the code.

Dependency inversion principle

Dependency inversion principle

There are many ways to express the DIP:

- Abstractions should not depend on details
- Code should depend on things that are at the same or higher level of abstraction

Branch by abstraction

Branch by abstraction

Is a technique for making a large-scale change to a software system

in gradual way that allows you to release the system regularly while the change is still in-progress.

Technical debt

Technical debt

A concept that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer. If technical debt is not repaid, it can accumulate 'interest', making it harder to implement changes later on. Technical debt is not necessarily a bad thing, and sometimes is required to move projects forward.

Parallel change

Parallel change

Parallel change, also known as expand and contract, is a pattern to implement backward-incompatible changes to an interface in a safe manner, by breaking the change into three distinct phases: expand, migrate, and contract.

Feature toggle

Feature toggle

Is a technique in software development that attempts to provide an alternative to maintaining multiple source-code branches (known as feature branches), such that a feature can be tested even before it is completed and ready for release. Feature toggle is used to hide, enable or disable the feature during run time.

Walking skeleton

Walking skeleton

Is a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel.

