Common Program Layouts

Flat

This is the simplest structure to start with and is the most common when you are starting with an application, only have a small number of files, and are still learning about the requirements. It's much easier to evolve a flat structure into a modular structure, so it's best to keep it simple at the start and partition it out later as the project grows.

- Pros:
    - It's great for small applications and libraries
    - There are no circular dependencies
    - It's easy to refactor into a modular structure
- Cons:
    - This can be complex and disorganized as the project grows
    - Everything can be accessed and modified by everything else

Grouping code by function

Code is separated by its similar functionality. In a Go REST API project, as an example, Go files are commonly grouped by handlers and models.

- Pros:
    - It's easy to refactor your code into other modular structures
    - It's easy to organize
    - It discourages a global state
- Cons:
    - Shared variables or functionality may not have a clear place to live
    - It can be unclear where initialization occurs

Grouping by module

Unfortunately, the title of this style of architecture is a bit redundant. To clarify, grouping by module means creating individual packages that each serve a function and contain everything necessary to accomplish these functions within them:

- Pros:
    - It's easier to maintain
    - There is faster development
    - There is low coupling and high cohesion
- Cons:
    - It's complex and harder to understand
    - It must have strict rules to remain well organized
    - It may cause stuttering in package method names
    - It can be unclear how to organize aggregated functionality
    - Circular dependencies may occur

Grouping by context

This type of structure is typically driven by the domain or the specific subject for which the project is being developed. The common domain language used in communication between developers and domain experts is often referred to as a ubiquitous language. It helps developers to understand the business and helps domain experts to understand the technical impact of changes.

Hexagonal architecture, also called ports and adapters, is a popular domain-driven design architecture that conceptually divides the functional areas of an application across multiple layers. The boundaries between these layers are interfaces, also called ports, which define how they communicate with each other, and the adapters that exist between the layers. In this layered architecture, the outer layers can only talk to the inner layers, not the other way around:

- Pros:
    - There is increased communication between members of the business team and developers
    - It’s flexible as business requirements change
    - It’s easy to maintain
- Cons:
    - It requires domain expertise and for developers to understand the business first before implementation
    - It’s costly since it requires longer initial development times
    - It’s not suited to short-term projects