Modular Programming

The concept of modular programming originated in the 1960s to help users. Programmers began to divide the more extensive programs into smaller parts. Though the concept of modular programming is decades old, it is the most convenient programming methods.

Definition

Modular programming is defined as a software design technique that focuses on separating the program functionality into independent, interchangeable methods/modules. Each of them contains everything needed to execute only one aspect of functionality.

Talking of modularity in terms of files and repositories, modularity can be on different levels.

- Libraries in projects
- Function in the files
- Files in the libraries or repositories

Modularity is all about making blocks, and each block is made with the help of other blocks.

Every block is solid and testable and can be stacked together to create an entire application. Therefore, thinking about the concept of modularity is also like building the whole architecture of the application.

Module

A module is defined as a part of a software program that contains one or more routines. When we merge one or more modules, it makes up a program. Whenever a product is built on an enterprise level, it is a built-in module, and each module performs different operations and business.

Modules are implemented in the program through interfaces. The introduction of modularity allowed programmers to reuse prewritten code with new applications.

Modules are created and merged with compilers, in which each module performs a business or routine operation within the program.

For example – SAP (System, Applications, and Products) comprises large modules like finance, payroll, supply chain, etc. In terms of software example of a module is Microsoft Word which uses Microsoft paint to help users create drawings and paintings.

Advantages of modular programming

- Code is easier to read - Working on modular programming makes code easier to read because functions perform different tasks as compared to monolithic codes. Sometimes modular programming can be a bit messy if we pass arguments and variables in different functions. The use of modules should be done in a sensible manner so as to avoid any problem. Functions should be neat, clean, and descriptive.
- Code is easier to test - In software, some functions perform fewer tasks and also functions that perform numerous tasks. If the software is easily split using modules, it becomes easier to test. We can also focus on the riskier functions during testing and need more test cases to make it bug-free.
- Reusability - There are times where a piece of code is implemented everywhere in our program. Instead of copying and pasting it, again and again, modularity gives us the advantage of reusability so that we can pull our code from anywhere using interfaces or libraries. The concept of reusability also reduces the size of our program.
- Faster fixes - Suppose there is an error in the payment options in any application, and the bug needs to be removed. Modularity can be a great help because we know that there will be a separate function that will contain the code of payments, and only that function will only be rectified. Thus using modules to find and fixing bugs becomes much more smooth and maintainable.
- Low-risk update - In modular programming, a defined layer of APIs protects things that use it from making changes inside the library. Unless there is a change in the API, there is a low risk for someone's code-breaking. For example, if you didn't have explicit APIs and someone changed a function they thought was only used within that same library (but it was used elsewhere), they could accidentally break something.
- Easy collaboration - Different developers work on a single piece of code in the team. There are chances of conflicts when there's a git merge. This conflict can be reduced if the code is split between more functions, files, repos, etc. We can also provide ownership to specific code modules, where a team member can break them down into smaller tasks.

Disadvantages of modular programming

- There is a need for extra time and budget for a product in modular programming.
- It is a challenging task to combine all the modules.
- Careful documentation is required so that other program modules are not affected.
- Some modules may partly repeat the task performed by other modules. Hence, Modular programs need more memory space and extra time for execution.
- Integrating various modules into a single program may not be a task because different people working on the design of different modules may not have the same style.

- It reduces the program's efficiency because testing and debugging are time-consuming, where each function contains a thousand lines of code.

Modular Application Development

The process of building a modular app starts with breaking down the various functionalities of your app into separate modules. There are two approaches that most organizations use to do this - modularizing by feature and modularizing by layer - but the one you should use really depends on your team and the project you're working on.

Modularize by Feature

In this modularization approach, developers break down code based on different app features, and it creates more modules as each module is defining a unique feature. Later, they integrate those modules to make communication between them easier and have an application that fulfils the required results.

Modularizing code by features has the following key characteristics.

- It's the best approach when you have a large project or are working with a lot of different team members.
- This approach offers high-level modularization.
- It enhances the reusability of modules.
- If not executed properly, it can occasionally make it more challenging for modules to communicate with one another.
- This approach has very strict development and deployment rules.

Modularize by Layer

In this approach, all the feature modules come together under a single domain and have more flexible rules. This approach helps developers use separate modules for different features as well.

Modularizing code by layers has the following key characteristics.

- This approach leads to fewer isolated modules and makes management easier.
- Only layers are reusable for multiple apps.
- It has a resemblance to monolithic applications.

Designing Modular Applications

Modular application development requires a proper understanding of application features and modular architecture. If you are struggling to determine the best way to build a modular app for your project, the following tips may be helpful to you.

Understand Project Details

Before you do anything else, make sure that you and everyone on your team have a solid understanding of the project before you move into the development phase. As modular structure revolves around features and layers, having a better understanding of the different features of your app is vital to breaking projects into multiple modules. It's always easier to code, test, and integrate different modules if you start out on the right foot and break code into modules wisely.

Keep Your Code Simple

Keep your code simple and focus on one thing at a time before moving on. Each snippet of code should perform a single task within the scope of your project. If you find yourself adding more tasks to your codebase without breaking them down into separate modules, it may be time to refactor it.

Use Abstract Interfaces

Abstract interfaces promote seamless communication between different segments of code. It's important that you come up with an abstract interface for each module so that it can interact with other modules through well-defined interfaces rather than concrete implementations. This makes it easier for developers on your team to modify or add new features without breaking existing functionality.

DRY Coding

Keep your code DRY, or Don't Repeat Yourself. DRY is a key principle in software development because it ensures that no piece of information is stored twice in different parts of the program's source code.

- This reduces redundancy which makes programs easier to read and understand as well as more maintainable over time.
- To avoid repetition in your codebase, use functions instead of duplicating lines of code everywhere in your project's source files.
- Also, whenever possible use objects instead of primitive values or constants like strings or numbers when declaring variables for better reusability and extensibility in future updates/versions of your app. This makes your code more reusable and scalable!

Use Names that are Easy to Understand

When you are working on multiple modules, it can be easy to mix up different module names, so provide all your modules with clear and easy-to-understand names. This way, any developer looking at your code will know what it does just by glancing at its name.