Choosing between object-oriented programming (OOP) and functional programming (FP) can significantly impact a SaaS business, especially in terms of product development, scalability, maintainability, and overall cost-effectiveness. While modular programming is not considered a paradigm on the same level as OOP or FP, it remains an essential aspect of code organization and understanding in software projects. I will explain those three elements and their impact on long-term SaaS business.

Object-oriented programming vs functional programming

Since OOP and FP layer is functioning on the same level, it will be easiest to understand these concepts by comparing them.

What is functional programming?

As one of the oldest programming paradigms, functional programming builds the systems by focusing on their functions and using them to manipulate data without changing the data itself. Imagine it as a pipeline where data flows through various steps (functions) without risking accidental changes to data along the way. It’s useful for building systems that are predictable and easy to test.

Computation in FP is based on evaluating mathematical functions using pure functions (no side effects), immutability (data cannot be modified), and higher-order functions (functions that accept or return other functions). In FP, functions are first-class citizens (because they are base of all operations), which means they can be assigned to variables, passed as arguments, or returned by other functions. The function is predictable: given the same input, it will always return the same output.

Example functions can serve purposes such as:

- suming/deducting and all other mathematical operations made on passed arguments
- storing a variable on drive or on database
- passing as an argument to another function and/or returning data from another function

For this article, we will only go into a few technical details. However, it is safe to say

that functional programming is great for the server side of software, data manipulation, or web crawling.

What is object-oriented programming?

Object-oriented programming (OOP) uses code to achieve the desired result step-by-step. OOP organizes software around real-world ‘objects,’ each with unique attributes (data) and actions (methods) that define its behavior.. Think of it as creating a blueprint for a product with details like color, shape, and actions it can perform. This approach helps build software that’s modular and reusable.

Software design in OOP is organized around data, or objects, rather than functions and logic. OOP principles include encapsulation (hiding data inside objects), inheritance (sharing attributes and methods with sub-objects), and polymorphism (allowing different objects to be treated as instances of the same type).

To better describe how objects work, let’s take a “phone” as an example of an object:

- Object: phone
- Functions (methods in OOP): calling, sending text messages, use of applications
- Variables (properties in OOP): color, make, model

Essentially, OOP is ideal for systems requiring many instances of the same object that behave differently based on unique parameters and relationships.

Modular programming – an old paradigm or a modern solution?

Modular programming is an alternative not strictly belonging to any of the methods mentioned above (although many claim it is one of the object-oriented programming methods). This method has been present in software development since the 1960s. Yet, it is still considered one of the most convenient programming methods. Modular programming is closely related to object-oriented and functional programming. They all have the same aim of dividing the construction of large software systems into smaller pieces.

In modular programming, a given software's functionality gets divided into manageable independent parts. In other words, the developer closes all elements of one functionality in one module. Independent modules can be bound together and stacked to form the whole application. You may be more familiar by name "library" because sometimes a library imported into system is (on programming level or on concept level) single module. So let's look closely at what modules are.

What are "modules" in programming?

Modules encapsulate specific functionality, performing distinct operations that, when combined, enable comprehensive software development. This approach is beneficial for producing many applications mainly because it allows for the preproduction of certain functions and reusing them to create new software. How exactly? Let's take an example of a notebook. For instance, in a plain lined notebook, the interior pages are standard, regardless of the notebook's cover design. The only element that changes is the cover. A print house could easily have one saved file called Lined Interior and would not have to create a new interior each time they want to release a new lined notebook. What a save of the most crucial resources – time and money! It is a simplified representation of how reusing prewritten code works and is direct implementation of DRY (Do Not Repeat Yourself) method, which is a basic rule for any good developer.

How can modules help in software development?

Readability

Thanks to module programming, the code is straightforward to read. However, it is noteworthy that modules can be long and complex, with many arguments and variables. Therefore it takes an experienced developer to conduct straightforward and clear modules. Nevertheless, if performed correctly, the modules should be easier to read than monolithic codes.

Easy testing and faster fixes

Every code needs to undergo testing before the software can be released. After all, the end product must work. Module programming allows testing separate parts of the whole software. It saves time (again) and ultimately makes it easier to perform the tests. It also allows one to focus on more complex functions that need more attention during the testing phase. Also, once a faulty module is identified, it is

simple to rewrite or replace just that one defective module, not the entire application.

Reusability

This function of module programming has already been briefly mentioned. However, it has more aspects worth mentioning. Firstly, thanks to modules, which can be pulled by using interfaces or libraries, the maximum size of created application or software is smaller than what it would be without using modules. Secondly, modules can be used within one piece of software and in creating other projects.

Low-risk updates

You might be wondering about issues that may come with updates. What if you make a faulty change in the module used in many more places or even applications? It could break the code and cause more havoc than the monolithic codes. It is a valid concern. However, there is more to a programming module than simply populating it. Every module programming project has a defined layer of APIs protecting from making changes inside libraries. Therefore, without changing APIs, there is a low risk of accidentally breaking the code.

Manageability

There are few aspects of this element. One is mentioned before testability. Second would be replaceability – meaning that if compatible, module can be replaced with another module or it's better version without affecting whole application. Third aspect would be easier coding – developer can focus on working only on specific module, without considering whole application at all times – he only needs to consider inputs and outputs of given module.

Are there any cons to modular programming?

Every method has disadvantages, and it would not be objective if we skipped them. That is why let us briefly explain the possible obstacles in creating modular software:

Proper skill is required

Programming modules requires skilled professionals, making it both time-consuming and potentially costly to find qualified developers or companies to

implement modular structures effectively. On the other hand, skilled developer will be much more effective when using modular programming for larger systems.

Need time for testing

As much as programming modules usually saves time, it may be time-consuming to test them. This is because you need to test many instances of the same object with different parametrs, so there are much more different mutations possible that need to be tested out.

Complexity control

Complexity increases if there are too many modules or if modules are too parametrized – so there needs to be balance between those two. Although this can be scaled up for bigger applications by using tree-like approach, or “silosing” modules to specific teams and reduce communication between them to APIs.

Overhead for communication

There can exist performance overhead, as additional memory and CPU time need to be used to communicate between modules. While some languages allow to compile multiple modules into single instance, still compilers aren’t prefect and other languages – especially script languages – are often not compiled. Even though, such overhead will only be visible for heavy scripts such as games.

Verdict for modular programming

As you can easly see, it is no-brainer: whenever possible, if developer skills allows it, he should use modular programming. Even though modular programming is not a perfect method, it is still a prevalent approach among many developers. Any disadvantage can be mitigated if one learns the programming module art properly. Its long-term impact is beneficial for both developers and business efficiency. Only thing to keep in mind is – as always – not overdo it, but find balance between complexity of modules and their amount, and if manageability drops – organize them in silos.

Object-oriented and functional programming

impact on business

Choosing between object-oriented programming (OOP) and functional programming (FP) can significantly impact a SaaS business, especially in terms of product development, scalability, maintainability, and overall cost-effectiveness.

Development Speed and Team Productivity

Object-oriented design is often more intuitive for teams familiar with creating complex, stateful applications like many SaaS solutions. It allows encapsulation and organization around “real-world” entities (like users, accounts, or projects), which can simplify the mapping of business logic. For teams well-versed in OOP, this may mean faster initial development. On the other hand, functional programming emphasizes immutability and pure functions, which can help reduce bugs and make the code more predictable and easier to test. If the development team is experienced with FP languages (like Haskell, Elixir, or Scala), they might actually produce more reliable code more quickly, but the learning curve can be steep for newcomers, impacting onboarding speed.

Scalability and Performance

SaaS applications developed in OOP languages often use mutable states, which can lead to performance bottlenecks as the number of users scales, especially if the architecture isn’t carefully designed. State management can become challenging as the application grows, and sometimes state tracking can lead to large history log files. For functional programming, since it emphases on immutability and stateless functions, can make scaling more straightforward. For instance, serverless architectures, microservices, or event-driven systems, common in SaaS, align well with FP principles. FP can better support concurrency and parallelism, helping improve performance as user demand increases.

Code Maintainability and Readability

While OOP can create readable and intuitive structures for representing complex entities, large OOP-based codebases may become difficult to maintain over time due to tight coupling, deep inheritance hierarchies, or complex interactions between classes. This can make it challenging to introduce new features or refactor code, affecting long-term maintainability. Functional code on the other hand,

focused on pure functions, often minimizes dependencies between components, making it easier to isolate and refactor different parts of the code. However, the code might be less readable to those unfamiliar with FP paradigms, especially in cases of heavy use of recursion or function composition. This can make maintenance harder if the team lacks functional programming expertise.

Testing and Debugging

Object-oriented code often involves more mutable state and side effects, making it harder to test, as functions depend on specific object states. Testing strategies like mocking and dependency injection can help, but they add complexity. You need to forsee many different combinations of many different parameters to cover everything as required. FP's reliance on pure functions—where outputs depend only on inputs and not on external state—can make automated testing simpler and more reliable. FP reduces side effects, which makes debugging easier since there are fewer "moving parts" in the code. For SaaS applications where reliability is key, this can mean fewer bugs reaching production.

Flexibility in Adding New Features

OOP is generally flexible for building new features if the application is well-designed, as you can extend classes or create new ones to represent new entities. However, modifying or extending deeply nested or complex hierarchies can introduce regression bugs. Adding features in FP might involve creating new functions or composing existing ones. While this can make for highly modular and composable code, complex business requirements can sometimes be harder to model purely in FP, depending on the language and how strictly it enforces FP principles.

Cost Implications (Infrastructure and Talent)

- **Infrastructure Costs**: FP languages often perform well in distributed, cloud-based, and serverless environments, which are common in SaaS, potentially reducing infrastructure costs through optimized resource utilization. OOP languages, especially those that support heavy multi-threading (e.g., Java, C#), can also scale well but may require more resources if state management becomes complex. Although one might argue that infrastructure cost is more dependent on framework used for specific language.
- **Talent Costs**: OOP is more common, so finding developers skilled in OOP

languages is generally easier and potentially less costly. FP experts, especially those skilled in languages like Haskell or Erlang, are in shorter supply, potentially increasing hiring costs. But again one might argue, that it is more dependent on language itself.

Adaptability to SaaS Requirements (Microservices, Event-Driven Architectures)

SaaS applications with complex, business-specific objects may benefit from OOP, especially if the architecture is built around microservices. Each service could represent distinct entities, with object-oriented design mapping directly to those entities. On the other hand functional programming can be particularly powerful in microservices or event-driven architectures, where each service processes data flows in stateless, isolated ways. FP's emphasis on immutability and statelessness aligns well with these patterns, allowing for resilient and independently deployable services that are easy to update or scale.

Decision Summary: Object-Oriented vs. Functional Programming in SaaS

Aspect	**Functional Programming (FP)**	**Object-Oriented Programming (OOP)**
Data Handling	Immutable data and pure functions	Objects with encapsulated data and state
Focus	Functions and transformations of data	Modeling real-world entities as objects
Reusability	Functions are reusable and composable	Classes and objects are reusable and modular
Ease of Testing	Often easier due to stateless and predictable functions	Testing depends on well-defined class interfaces and encapsulation

In sum, object-oriented programming tends to be more beneficial when:

- Business requirements involve complex, interconnected data structures.
- The development team has strong experience in OOP.
- There is a focus on quick onboarding and straightforward team scaling.

On the other hand, functional programming may offer an advantage when:

- The application is expected to scale horizontally with high concurrency.
- Reliability and simplicity in testing and maintenance are high priorities.
- The team has FP expertise, and there is room for a learning curve if needed.

Ultimately, many modern SaaS projects adopt a **hybrid approach**: leveraging OOP for areas requiring structured data representation and FP for stateless services or components where concurrency and fault tolerance are critical. This balance can often capture the strengths of both paradigms and meet SaaS requirements effectively. You may be able to identify which elemetns are OOP and which are FP on tech descripiton of your application, but it is probably just best to leave it developers to decide if you are not application architect. What should be more important if you want to run SaaS is to create perfect lean canvas or business canvas model which will help developers a lot in understanding your application goal and business strategy, which ultimately will lead to crafting real bespoke solution.

Sailing Byte has been using modules for years

With countless advantages of modular programming and our extensive experience in the software development market, we can safely say that Sailing Byte is an expert in module programming. We design modules in advance and plan what to include in every one of them so they are well-written and as simple as possible.Finally, we reuse models creating our style of developing high-quality software perfectly tailored to our clients' business needs. Book a call today to find out about all possibilities of modular programming we can offer while building your software.