

introduction to software design: the principles

Tech Talk

Start



Introduction

Software design is the process of envisioning and defining the architecture, components, modules, interfaces, and data of a system to meet defined requirements. Its purpose is to create a detailed plan or blueprint for the software system, which guides developers during implementation. This ensures the software meets user needs and can be effectively maintained and improved over time.



software design principles

you can recognize a good doctor if he fully and wholeheartedly respect and follow the Hippocratic Oath.

You can tell Eikichi Onizuka is a great teacher because he follows fully and wholeheartedly, the Socratic Oath.

In the same logic, you can spot a skilled software engineer by his knowledge and practice of software design principles in his work.






Abstraction and encapsulation



● ● ● Abstraction

Abstraction in software design simplifies complex ideas by focusing on what's important and leaving out unnecessary details. It helps developers work with higher-level concepts like data types or interfaces without needing to know every technical detail underneath. For example, when you use a variable in code, you're abstracted from how it's stored in memory, allowing you to focus on using it effectively.



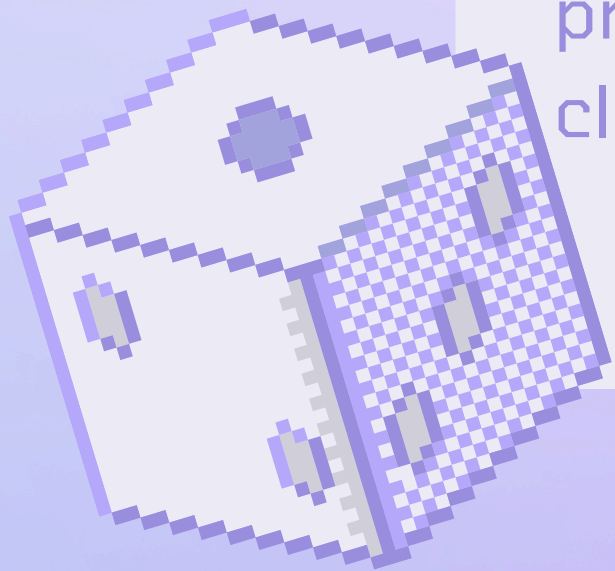
● ● ● encapsulation

Encapsulation complements abstraction by grouping data and methods that work on that data into a single unit. This unit shields the inner workings from outside interference, ensuring that data is accessed and modified only through defined interfaces. This helps maintain code integrity and makes it easier to manage and extend software projects over time.



YAGNI

When designing a solution, it's important to balance making it adaptable to the current system and keeping it simple. Adding features too early for future needs can make the solution unnecessarily complex and harder to maintain. This goes against the "You Aren't Gonna Need It" (YAGNI) principle, which advises against adding functionality until it's actually needed. By focusing on solving the present problem effectively and avoiding premature additions, developers can ensure their code remains clear, efficient, and easier to manage in the long run.



DRY and KISS principles

DRY

Don't repeat yourself, duplication in code can lead to inconsistencies and bugs. If you duplicate your logic in several places and then find a bug, you're more likely to forget to change it somewhere else in your code. Instead, find a repetitive functionality, abstract it in the form of a procedure, class, etc., give it a meaningful name and use it where needed.

KISS

Keep it simple, stupid. Keeping code as simple and straightforward as possible, as simple code is easier to understand and maintain and less prone to errors. The main idea behind it is to focus on the simplicity of a system, which increases understanding and reduces overengineering while using only the tools you really need.

Law of Demeter and separation of concerns

LoD

The Law of Demeter (LoD), also known as the principle of least knowledge, emphasizes limiting direct interactions between objects. In OOP, LoD suggests that an object should only communicate with its immediate "friends" — other objects it directly interacts with, such as those it creates, owns, or receives as method parameters. Avoiding interactions with "strangers" — objects not directly associated with the current one — enhances code readability and maintainability.

CoD

The Separation of Concerns (SoC) principle advises dividing a system into smaller parts, each addressing specific responsibilities independently. This approach ensures clarity, modularity, and reusability in software design. Similar to the abstraction principle, SoC helps create systems with well-defined interfaces, making them easier to understand, maintain, and extend over time. By separating concerns effectively, developers can manage complexity and build robust software architectures that adapt to evolving needs efficiently.



Conclusion

Due to poor time management, This is where my tech talk ends. There's no conclusion. I hope this was somewhat interesting, Insightful and clear. Thank you for your time !

