More Design Principles

Kamonphop Srisopha

Kamonphop.s@cmu.ac.th

Faculty of Science, Chiang Mai University

คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

Agenda:

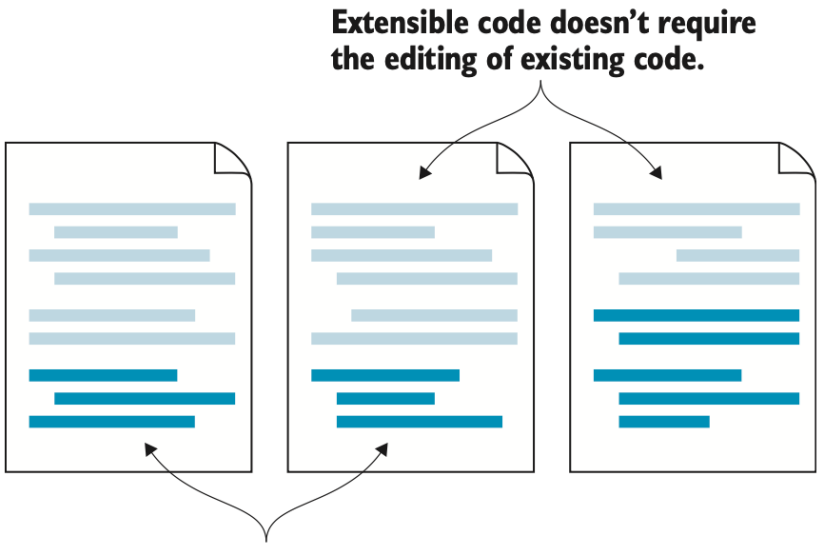
- More General Design Principles
- SOLID Principles

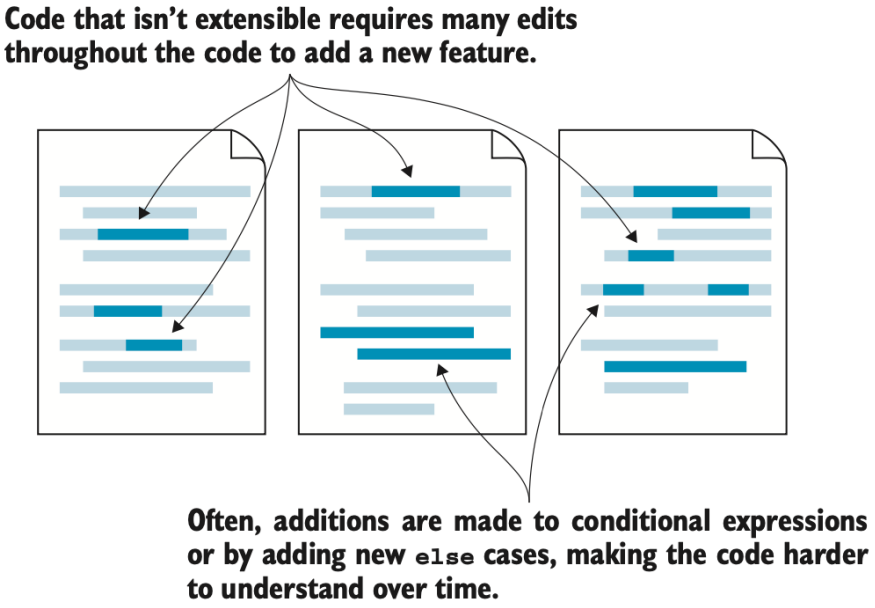
Goal

- Make things maintainable, flexible, reusable, and scalable
- When we work on your code, try to think “what can i do now if the problem changes in the future my code still works or I only need to do very minor adjustments to accomondate that”

เราต้องการที่จัดระเบียบส่วนประกอบ,โครงสร้าง, และความสัมพันธ์ต่าง ๆ ของระบบของเราให้ดีที่สุด เพื่อที่ลดผลกระทบจากการเปลี่ยนแปลง และทำให้ซอฟต์แวร์เรา maintain ได้ง่ายที่สุดเท่าที่จะทำได้

Maintainable and Extensible Code

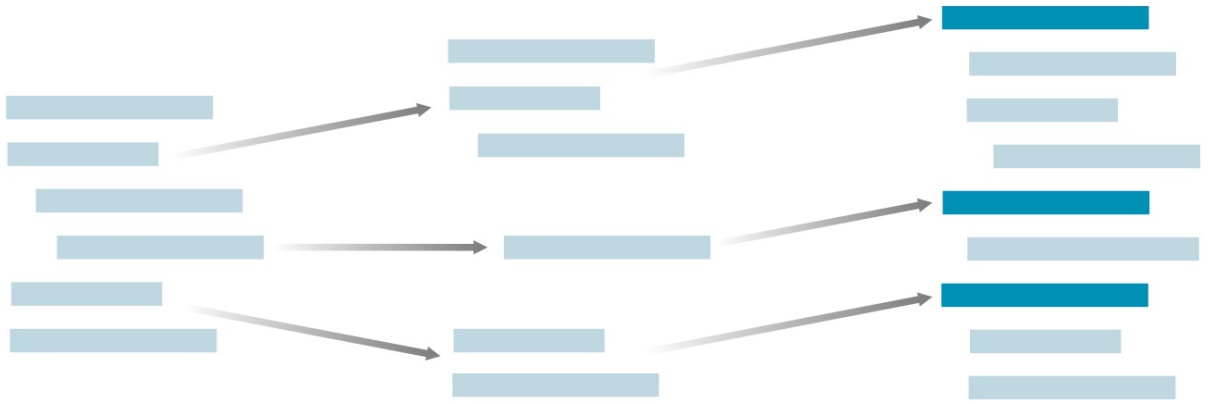




More Design Principles

Separation of Concerns

- Separating a program into distinct sections, where each section addresses a specific concern or part of the problem. This makes the code more organized and easier to manage.

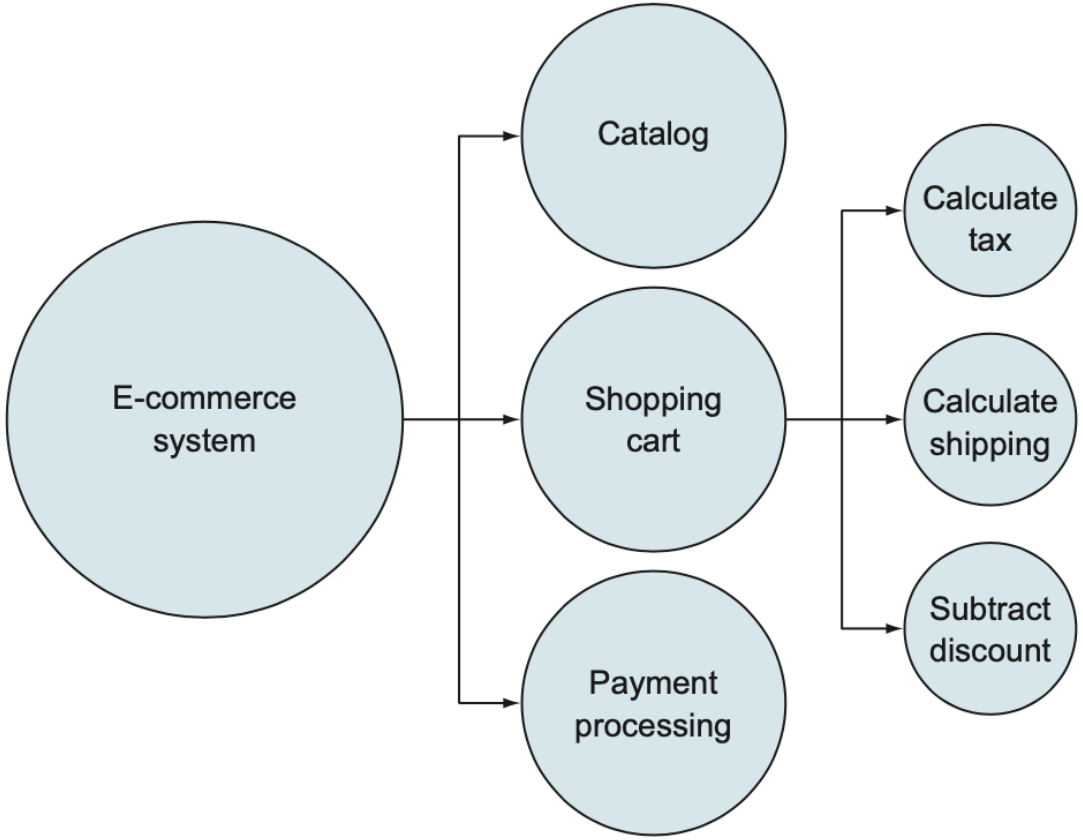


A long stretch of code can be difficult to understand. Nested conditionals and tasks spread out over many lines make it hard to follow.

Identify the unique tasks being executed in the code. Group together lines that calculate intermediate values with the lines that calculate the final result.

Wrap each task in a function with a name that clearly indicates what it does. Give each input argument a name that conveys its intent and, ideally, its data type.

Decomposition



This e-commerce system is all-encompassing and difficult to understand in full.

Breaking the system into its constituent phases helps give some shape to everything that's happening.

Breaking each phase into tasks results in actionable pieces of functionality.

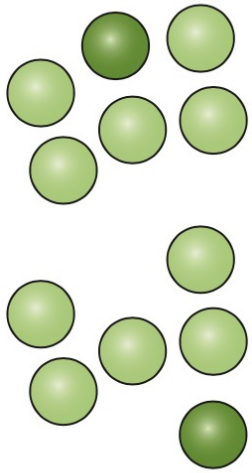
Increase Cohesion

- Ensure that elements within a group (like packages or classes) are related and fit together logically.

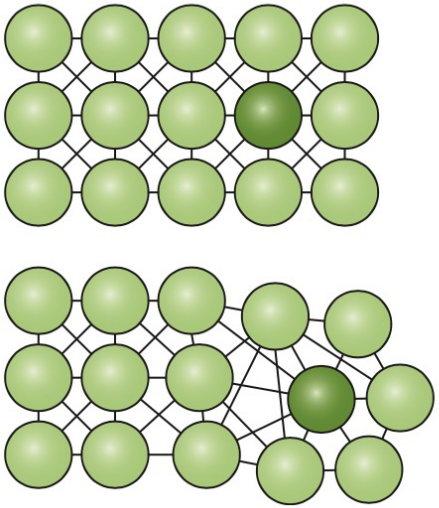
Reduce Coupling

- Minimize interdependencies between components to simplify debugging and modifications.
- Aim for independent components to avoid complex dependencies and ease troubleshooting.

Loosely coupled pieces of code can move around and change their shape freely, just like the molecules in a liquid.



Tightly coupled pieces of code rely on the code around them. Changing one piece is difficult because the other pieces must move to accommodate it.



Increase Abstraction

- Generalize components instead of focusing on specific details.
- Write code that is general and abstract, allowing for its use in various contexts and reducing the need for rewriting. – **For greater flexibility**

Error-Proof Your Code

- Write robust, error-resistant code with informative error messages.
- Assume users might misuse the code and design to handle such scenarios effectively.

Law of Demeter
(principle of least knowledge)

• A module should only know about its immediate dependencies and not the internal details of other modules

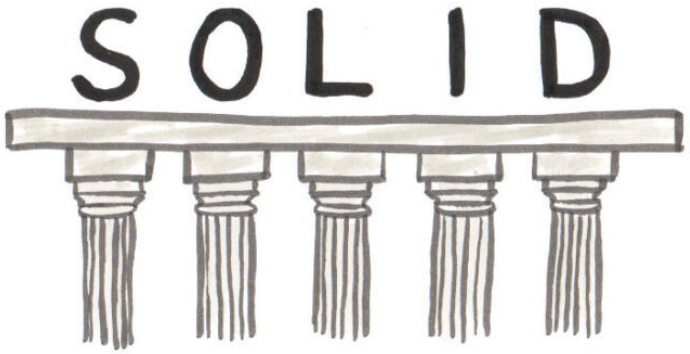
You Aren’t Gonna Need It

• Avoid implementing features or functionalities that aren’t currently needed, as it can lead to wasted effort and complex code.

SOLID Principles

SOLID Principles

หลักการออกแบบ object-oriented class ให้เป็นแบบ Best Practice ที่นักพัฒนาโปรแกรมควรจะปฏิบัติตามหรือนำมาใช้ เมื่อทำการออกแบบ class ตอนเขียนโปรแกรม



- **S** — Single-Responsibility principle
- **O** — Open/Closed principle
- **L** — Liskov Substitution principle
- **I** — Interface Segregation principle
- **D** — Dependency Inversion principle

Single Responsibility Principle

There should never be more than one reason for a class to change

คลาสหนึ่งๆควรมีเหตุผลเดียวที่จะแก้มัน (คลาสควรทำงานแค่อย่างเดียว เราจะแก้มันเพราะต้องการแก้การทำงานเท่านั้น)

ถ้าคลาสทำงานหลายอย่างมันจะทำให้เราสับสนว่าหน้าที่จริงๆของมันคืออะไร ถ้ามันทำหลายอย่างมันจะทำให้เราเข้าใจมันยากและอาจจะเกิดปัญหาได้เมื่อเราแก้ไขมัน

Goal = High Cohesion (ความสอดคล้องกันภายในสูง)

Open-closed Principle

> Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification

เราควรจะสามารถเพิ่มการทำงานของ entity หนึ่งๆ โดยที่ไม่ต้องไปแตะต้องโค้ดใน entity นั้นๆ (เปิดต่อการเพิ่ม แต่ปิดต่อการแก้ไข)

Liskov Substitution Principle

> Objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program

Object ของคลาสแม่ควรจะสามารถถูกแทนที่ได้ด้วย object ของคลาสลูกได้โดยไม่กระทบการทำงานของตัวโปรแกรม (หรือ subclass ควรแทนที่ superclass ได้)

Interface Segregation Princinple

> “Clients should not be forced to implement interfaces they do not use”

Segregation = แบ่งแยกจากกัน

แบ่งแยก interface ออกตามการใช้งานให้ตรงจุด ดีกว่าสร้าง general interface เพื่อให้ทุกอย่างใช้งานร่วมกัน

Class (หรือผู้ใช้ clients) ไม่ควรที่จะถูกบังคับให้ต้อง implement function จาก interface ที่มันไม่ได้ใช้งาน

Dependency Inversion Principle

> Depend upon abstractions, not concretions.

- Entity ที่เป็น High level ไม่ควรไปผูกติดกับ Low level หากแต่ทั้งสองควรรู้จักกันในรูปแบบ abstraction (interface) เท่านั้น
- Abstraction ไม่ควรมีรายละเอียดการทำงาน แต่รายละเอียดการทำงานควรขึ้นกับ abstraction
- Goal = Decoupling (แยกการทำงานออกจากกัน)

Questions?

References

- “Practices of the Python Pro” by Dane Hillard