Design Pattern

- Yazılan kodun anlaşılabilir olması (class isimleri, method ve variable isimleri).

* GRASP → General Responsibility Assignment Software
  → Doing Responsibility ve Knowing Responsibility Patterns.

Doing Responsibility

- Class'ların sorumlulukları ve görevleri

Knowing Responsibility

- Kodun içeriğini bilmek. (encapsulation)

RDD (Responsibility Driven Design)

- Kullanılan objectlerin birlikte bir amaca hizmet etmeleri.

Five Grasp Patterns.

- Creator Pattern
- Information Expert Pattern or Principle
- Low Coupling Pattern
- High Cohesion Pattern
- Controller Design Pattern

1) Creater Pattern

Problem: Who create an instance of Class A.

Solution: Class B'ye, A Class'ı için instance yaratma sorumluluğunun verilmesi. (bu sorumlulukları halleden B'nin içindeki methodlar)

2) Information Expert

Problem: How to assign responsibilities to objects.

Solution: Calculate the GPA of a student which class responsible for that.

* Seperation of Concern Principles

Bir projede çok class kullanılabilir ama kullanılan classların içinde çok fazla method olmaması, olması gerekir. (Az ve öz şekilde class kullanımı)

3) Low Coupling (High coupling bunun tam tersi olmasını istemiyoruz)

Problem: How to reduce the impact of change and encourage re-use

Solution: Classların birbirine bağlı olabildiğince az tutmak. Bunun nedeni kodda bir şey değiştiğinde impactı minimumda tutmak.

4) High Cohesion Pattern

High Cohesion = Low Coupling

- Cohesion: Measures how strongly related and focused are the responsibilities of an element. (Low cohesion → All responsibilities appointed to one class)
- Problem: How to keep classes focused and manageable
- Solution: Assign responsibility so that cohesion remains high

* Class with high cohesion has low number of methods with highly related functionality and doesn't do much work.

5) Controller Design Pattern

Problem: Who should be responsible for UI events?

Solution: Assign responsibility for receiving or handling system event in one of two ways.

Model View Seperation Principle

- UI objects should not contain application or business logic.

Summary: We should use all 5 Grasp Patterns Because reduce the impact of the change and easier to understand and maintainable OO code.
↓ Object oriented.

More Grasp Pattern:

Polymorphism → Giving same name to services different objects.
Different object types usually implement a common interface or common superclass

- Pluggable system → tak çıkart yapmalı kolay olmalı

Pure Fabrication

Problem: What object should have the responsibility when you do not want to violate high cohesion and low coupling, or other goals but solutions offered by expert are not appropriate?

Solution: Create new class to have that method — High cohesion
— Low coupling
They both does not violated

Analysis → Understanding the requirements of the customer
Design → About software, solutions for analysis
Code → Coding according to analysis and design phases.
Testing → Must do again and again until the customer satisfies.

RDD (Responsibility Driven Design)
TDD (Test Driven Design)

Unit Testing → it is about methods
State of an object is current values of attributes.

Test Driven Programming

- Before writing the actual code must create the units. XP (extreme programming)
  ↓ methods.
- Testing is done when you run out of time or money.

Testing OO Code

- Class Tests → Testing the methods of the class
- Integration Test
- Validation Test
- System Test

} These must be done in these order.

Class (Unit) Testing

- Smallest testable unit is encapsulated class.

* Challenge of the class testing → Encapsulation, Inheritance, Polymorphism
  White Box Test.

Integration Testing

1) Thread based testing → integrate classes required to respond to one input or event.
2) Use based testing → Integrate classes required by one use case.
3) Cluster Testing → Integrate classes required to demonstrate one collaboration.

Validation Testing

- If the simulation behaves as we expected it satisfies the customer.
- Testing is about customer's need.

* Alpha: at developer side * Beta: at customer's side

System Testing

- Finger pointing defence:
  1) Design error-handling paths that test external information.
  2) Conduct a series of tests that simulate bad data.
  3) Record the results of tests to use as an evidence.

Types of System Testing.

- Recovery testing → how well and fast your system recovers from its faults.
- Security Testing → Protection from hackers. (unauthorized access)
- Stress Testing → To see the limits of your program.
- performance Testing → Run time performance.

PYTHON

* Advantages → multiplatform support
  → It's quick for small projects.
* Disadvantages → slower than the compiler code.
- Strong typing → you cannot concatenate types.
- Dynamic typing → (Especially for python) variable'lerin tipi runtime'da belirlenir.

Sequence Data Types

- Strings → static - immutable
- Lists → dynamic - mutable
- Tupples → static - immutable

→ Once you created, you cannot change them.

* Unicode → ü, ş, ö, ı, ğ → Leri kullanmanı sağlar.
* Python is good for certain domains. → Data Science / Data mining Big Data, Artificial Intelligence (Machine learning)
* Lists are not array. Lists can be extended.
* Lists are not linkedlist either.
* Set → mutable, frozen sets are immutable.
* "()" tuple, "{}" dict, "[]" list

* Hashable Items.

- All immutable data types are hashable.

OOP in Python

- Everything is public. There are no private keyword in python but if an attribute like this. __name → this treated as private.

Logging

- Cyber security issues (log your system to avoid any errors or security violations)
- Debug → for development.
- → Info → expected behaviour
- Warning → there might be something unexpected
- Error → // // // // // is
  ↘ and software is not working.
- Critical → The software is not running anymore.

Automated Testing

- Unit test is good for testing.

* Testing should start in the beginning)