Design Patterns

\* What is the purpose of design patterns

* A design pattern “…names, abstracts, and identifies the key aspects of a common design structural that make it useful for creating a reusable object-oriented design.”\* A design pattern is a proven solution to a recurrent problem in a context. An effective, reusable, proven structure/communication solution for a given object-oriented design problem.

\* Describe the differences between creational, structural, and behavioral patterns

* Creational Patterns ● Abstract the instantiation process ● Define classes to handle object creation
* Structural Patterns ● Concerned with how classes and objects are composed to form larger structures ● Describe ways to compose objects to realize new functionality
* Behavioral Patterns ● Concerned with algorithms, flow of control, and assignment of responsibilities between objects ● Describe how a group of objects cooperate to perform a task

\* Pick one design pattern that we talked about and describe it in detail.

* Singleton pattern Pattern Category: Creational Intent: ● Ensure a class only has one instance, and provide a global point of access to it. Problem addressed: ● Ensuring that a class is instantiated only once, and that the resulting object is readily accessible. Solution: ● Make the class itself responsible for instantiation and knowing whether it has been instantiated.

Anti-Patterns

\* What is an anti-pattern? Why is this a problem?

* refer informally to any commonly reinvented but bad solution to a problem.

\* Pick two anti-patterns and describe what they are and why we want to avoid them.

* [Analysis paralysis](https://en.wikipedia.org/wiki/Analysis_paralysis): A project that has stalled in the analysis phase of development, and is unable to achieve support for any of the potential plans of its implementation
* [Bleeding edge](https://en.wikipedia.org/wiki/Bleeding_edge): Operating with cutting-edge technologies that are still untested or unstable, leading to cost overruns, under-performance or delayed delivery of the product

OOP/OOD/OOA

\* What are the high-level differences between OOP, OOA, and OOD? Are they all necessary?

* Object-Oriented Analysis (OOA) is concerned with: ● Developing an object-oriented model of the application domain. ● Identification of objects/entities and operations associated with the problem.
* Object-Oriented Design (OOD) is concerned with: ● Developing an object-oriented model of the system to implement requirements ● Implementing the solution by adding new objects to the ones already identified on the OOA phase
* Object-Oriented Programming (OOP) is concerned with: ● Realizing an OOD using an object-oriented programming language. ● Identifying additional objects that are language or API specific and necessary to implement the solution.

\* Why would one program to an interface?

* Object Interfaces ● Program to an interface, not an implementation
* Advantages: ● Clients are unaware of the specific class of the object they are using ● One object can be easily replaced by another ● Object connections need not be hardwired to an object of a specific class (increased flexibility) ● Loosens coupling ● Increases likelihood of reuse ● Improves opportunities for composition since contained objects can be of any class that implements a specific interface
* Disadvantages: ● Modest increase in design complexity

\* What are cohesion and coupling?

* Two software quality metrics were proposed (in the early 1970s by Stevens, Myers, and Constantine) for judging the quality of a module:
* Cohesion is the degree of relatedness/similarity between elements within a module. It is an intra-module measure. Modules with high cohesion tend to be more maintainable (i.e., modifiable and understandable) and reusable compared to modules with low cohesion.
* Coupling is the degree of dependence between two or more modules. \* Why are high cohesion and low coupling desirable?
* Functional Cohesion (Best / Most Desirable) Parts grouped to do one well-defined task
* Normal Coupling (Low / Most Desirable) ○ Data ○ Control

\* What are two methods for decomposing a problem? Describe an example of each.

* Object-Oriented decomposition: ● Assigns objects to module ● High-level design: ○ Identifies the system’s object types and explains how objects are related to one another ● Lower-level design: ○ Detail the objects’ attributes and operations
* Functional decomposition: ● Partitions functions or requirements into modules ● Begins with the functions that are listed in the requirements specification ● Lower-level designs divide these functions into sub-functions, which are then assigned to smaller modules ● Describes which modules (sub-functions) call each other

Test-Driven Development

\* What is the purpose of test-driven development? Why would you use it?

* The goal is to decrease the interval between writing tests and production code to a matter of a few minutes. Start (Green Light) ● Write a test ● Code may fail to compile (Yellow Light) ● Implement just enough (a stub) to compile ● Run the test and ensure it fails (Red Light) ● Implement just enough to make the test pass (Green Light)

\* Describe the steps of a test-driven development process.

* Green light 1) Create test Yellow light 2) Method doesn't exist a) Add stub to solve Red light 3) Method fails! Green light 4) Fix method

\* What is a function stub?

* ● Implement just enough (a stub) to compile. Method doesn't exist a) Add stub to solve. Stub: Dummy modules used to replace a module that is subordinate to (called by or used by) the module to be tested. A stub uses the subordinate modules interface, may do minimal data manipulation, and provides verification of entry and returns.

Verification & Validation

\* What is the key difference between verification and validation.

* Verification: A set of activities that ensure that the software conforms to its specifications: “Are we building the thing right?” Validation: A set of activities that ensure that the software built meets the customer’s needs and expectations. “Are we building the right thing?” Verification and Validation (V&V) should establish confidence that the software is “fit for purpose” → This does not mean the software is completely free of defects. → It does mean the software must be good enough for its intended use, where that use determines the degree of confidence that is needed.

\* Describe two approaches for performing integration testing.

* Non-incremental Integration: ● All modules are combined and tested as a whole ● Chaos usually results as a set of errors are encountered. ● Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. ● Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.
* Incremental Integration: ● Program is constructed and tested in small increments. ● Errors are easier to isolate and correct. ● Interfaces are more likely to be tested completely and repeatedly. ● Requires additional overhead in the form of drivers and/or stubs. ● Several approaches to incremental integration: ○ Top-down integration ○ Bottom-up integration ○ Sandwich integration

\* What is A/B testing? Give an example.

* Alpha Testing: ● Conducted at the developer’s site by end-users. ● Software is used with the developer “looking over the shoulder” of typical users and recording errors and usage problems. ● Conducted in a controlled environment.
* Beta Testing: ● Conducted at one or more end-user sites. ● Developer is generally not present. ● “Live” application of software in an environment that cannot be controlled by the developer. ● End-user records all problems (real or imagined) that are encountered during beta testing and reports them to the developer.

\* Is exhaustive testing typically feasible? Why or why not?

* Most common and least efficient method for isolating the cause of an error ● Hope to find a clue that can lead us to the cause of an error ● Using memory dumps, run-time traces, loading program with output statements.

\* What is a driver with respect to testing?

* Driver: In most applications, it is nothing more than a “main program” that accepts test case data, passes such data into the module to be tested, and prints the relevant results.

\* Be able to draw a flow graph, calculate complexity, and derive test cases for a function (i.e., what we did on the board)

Diagram

Description automatically generated

System-Building

\* What is the purpose of a build script?

* System building is the process of assembling software components into a program that executes on a particular target configuration

\* What are two common systems for building software

* Apache Ant ● MSBuild

\* What are two common tasks for building software

* Making sure the source directory contains your build
* Setting up java source directory properly to contain your build