Discussion 6 Object Oriented Programming (OOP)

Discussion Topic:

Object Oriented Programming (OOP), one of the most rapidly growing programming paradigms, typically uses cars or house blueprints as metaphors to explain the approach.
In this discussion, use the architecture of a house or building as a metaphor to draw comparisons with software architecture.

- How are the disciplines of traditional architecture and the software architecture similar?
- How are they different?

My Post:

Hello Class,

For context, Object Oriented Programming (OOP) is a programming paradigm that is the basis of programming languages such as Python, Java, and C++ (Mozilla, n.d.). The paradigm focuses on modeling a system as a collection of objects, where each object represents some particular aspect (e.g., a functionality, data structure, and behavior) of the system. Objects or classes the definitions of the object contain/define both functions (or methods) and data. An object is a code/data abstraction that provides an interface to other code (Module) that wants to use its functionality or data, while maintaining its own private, internal state.

Question-1: How are the disciplines of traditional architecture and software architecture similar?

When designing a house, traditional architects start designing the house at a high level of abstraction. They start by defining the house's "big picture." In other words, you wouldn't design a house by drawing plumbing first; you need a blueprint-level structure of the house before you can understand how to integrate/design plumbing for it. Similarly, software architects need a high-level representation of the system before they can design or integrate lower-level system modules. In object-oriented software architecture, this translates to identifying the major classes/components, the structure (hierarchies/aggregations), and the collaborations (messages) between them (Pressman & Maxim, 2020b).

Both architecture fields need to define a structure based on requirements and quality expectations of the project stakeholders. Traditional architects need to decide based on the project requirements, the structure style, type of materials, and size of the house; similarly, software architects need to define, based on identified requirements, the components, structure style, technology, and data structure of the software project. For OOP, this translates to defining classes, which are the "software building blocks" encapsulating the functionality, data, and behavior of the software.

Additionally, both architecture fields use templates to define structural styles. Traditional architects use "center hall colonial" guides house construction; similarly, software architectures follow recognizable styles (data-centered, layered, Object Oriented (OO), etc.) that define components, connectors, and constraints (Pressman & Maxim, 2020b). For OOP, this translates in terms of architectural styles such as a "layered" system or a Model-View-Controller (MVC) system.

Furthermore, both architecture fields range from high-level layout at first to detailed specifications. In other words, the software design model and the house design model transition from a high-level view of the project down to a "plumbing-level," high-detailed view (Pressman & Maxim, 2020a). For example:

1. Software architecture ⇔ house floor plan (view -> subsystems and relationships)
   ↓
2. Software interfaces ⇔ house doors/windows/utilities (view -> how information enters, exits, and moves inside)
   ↓
3. Software components ⇔ house wiring/plumbing of each room (internal data structures and algorithms)

In the context of OOP, this translates to moving from a high-level picture of class groupings based on their functionality or use (packages/subsystems) to the detailed definition of public method "interfaces" that control access, internal data structures/algorithms/private methods inside each class.

Question-2: How are they different?

The core difference resides in the software's capacity to be reshaped after implementation, compared to the house once it has been built. Especially software with OO architecture, due to their modular nature, changes can often be localized within a class or module, allowing these components to be easily adjusted, extended, or refactored (Pressman & Maxim, 2020a). On the one hand, a house is long-lasting, sturdier, and more resilient than software, which can slowly degrade due to technological advances, potentially become out-of-date, and not maintainable (technical debt, architectural erosion).

-Alex

References:

Mozilla. (n.d.). *Object-oriented programming*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Advanced_JavaScript_objects/Object-oriented_programming

Pressman, R. S., & Maxim, B. R. (2020a). Chapter 9 Design Concepts. Software engineering: A practitioner's approach (9th ed.). McGraw-Hill. ISBN: 9781260423297

Pressman, R. S., & Maxim, B. R. (2020b). Chapter 10: Architectural Design—A Recommended Approach. Software engineering: A practitioner's approach (9th ed.). McGraw-Hill. ISBN: 9781260423297