Chapter 14: Pattern-Based Design

January 27, 2021 9:21 AM

- Design Patterns
  - ⭐ A <u>design pattern</u> can be thought of as a three-part rule about a <u>context</u>, <u>problem</u>, and <u>solution</u>
  - Context allows the reader to understand the environment of the problem and the appropriate solution
  - ⭐ We use design patterns so that we don't have to "<u>re-invent the wheel</u>"

Effective design patterns

- Solves a problem =
  - patterns capture solutions, not just abstract principles or strategies
- Proven concept = patterns capture solutions with a proven track record, not theories or speculation
- Solution is not obvious =
  - Best patterns generate a solution to a problem indirectly
- Describe a relationship =
  - Patterns don't just describe modules but describe deeper system structures and mechanisms
- Elegant in it's a approach to unity

Kinds of patterns

- pg. 291
- Rubber necking
- creational patterns =
  - The "creation composition and representation of objects"
- Structural patterns =
  - Focuses on problems and solutions associated with how classes and objects are organized and integrated to build a larger structure
  - Behavioral patterns =
    - Responsibility between objects
- Architectural patterns =
  - Describes broad based design problems that are solved using a structural reproach
- Data patterns =
  - Describes recurring data
- Component patterns (design pattern) =
  - Address problems associated with the development of subsystems
- Interface design patterns =
  - Describe common user interface problems and their solutions within a system
- WebApp patterns =
  - Address a problem set that is encountered when building a WebApp
- Mobile patterns =
  - The context is a mobile platform

Frameworks

- Patterns themselves may not be sufficient to develop a complete design
- In cases it may be necessary to provide a implementation-specific skeletal infrastructure
- ⭐ You can select a <u>reusable architecture</u> that provides the generic structure and behavior for a family of software

A framework is not an architectural pattern, but rather a skeleton with a collection of plug points(hooks and slots)

Pattern description

- Pg.294
- Pattern name = describes the essence of the pattern in a short but expressive name
- Problem = describes the problem that the pattern address
- Motivation = provides an example of the program
- Context = describes the environment in which the problem resides
- Forces = list the system of forces that affect the manner In which the problem must be solved
- Solution = provides a detailed description of the solution properties
- Intent = describes the pattren and what it does
- Collaborations = describes how other patterns contribute to the solutions
- Consequences = describes the potential trade-offs that must be considered when a pattern is implemented
- Implementation = identifies special issues that should be considered when implementing the pattern
- Known uses = provides examples of actual uses of the design pattern in real applications
- Related patterns =

Pattern based design

- ★ A software designer begins with a requirements model (either explicit or implied) that presents an abstract representation of the system
- The requirements model describes the problem set, establishes context, and identifies the system forces
- If you discover you are faced with a problem, and system of forces that solved before, then use it
- If it is a new problem use methods and modeling to make a new pattern

Thinking in patterns

1. Be sure you understand the big picture (requirements model) / the context
2. Examining the big picture, extract the patterns that are present at the level of abstraction
3. Begin your design with big picture patterns that establish a context or skeleton for further design work
4. Work inward form the context
5. Repeat steps 1 - 4 until the design is completely fleshed out
6. Refine the design by adapting each pattern to the specifics of the software your trying to build

Design task pg.297 - 298

- Exam the requirements model and develop a hierarchy
- Determine if a reliable

Most pattern development is done from the top down rather then bottom up

Pattern organization table

- List patterns and systems

Common mistakes

- Not enough time being spent to understand the underlying problem , its context forces and as a consequence, you select the pattern that looks right but is actually wrong
- Once the wrong pattern is selected don't refuse to see the error and try to force it to work
- In other cases the problem has forces that are not considered by the pattern you have chosen, resulting in a poor erronious fit
- Sometimes a pattern is applied too literally and the required adaption for your problem

spaces are not implemented

Component- level-design

- Component-level design patterns provide a proven solution that addresses one or more sub-problems extracted from the requirements model
- In many cases, design patterns of this type focus on some functional element of a system
- Having enunciated the sub-problem that must be solved, consider context and the system of forces that affect the solution

Anti-patterns

- ★ Anti-patterns describe commonly used solutions to design problems that have a negative affect on the software
- Ant-patterns can provide tools to help developers recognize when these problems exist and may provide detailed plans for reversing the underlying problem causes and implementing

Selected anti-patterns pg.304

- Blob = single class with a large number of attributes, operators or both
- Stovepipe system = a barley maintainable assemblage of ill-related components
- ★ Boat anchor = retaining a part of the system that no longer has any use
- Spaghetti code = program whose structure is barley comprehensible, especially because of misuse code constraints
- Copy and paste programs
- ★ Silver bullet
- Programming by permutation