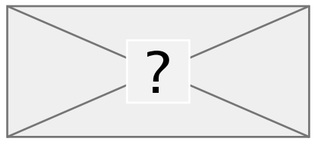
Large Scale Software Engineering

Week 05

Mike Helmick
University of Cincinnati
CS6028
Spring 2014

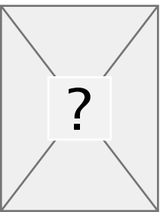


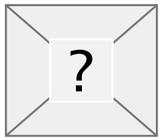
Chapter 11

Encapsulation and Partitioning

Partitioning

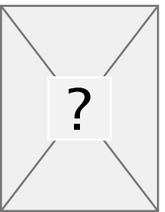
- No how you divide up the data / users in your system
- This is about how you partition your actual software
    - Components and how they interact with each other

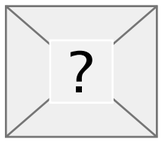




Layers

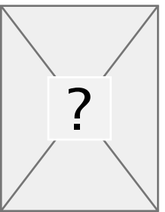
- Ideally, we want to construct software that is easy to understand
    - That makes it easy to modify

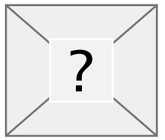




Creating Levels - Guidelines

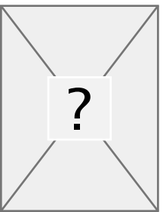
1. Create levels of abstraction through hierarchically nesting elements
2. Limit the number of elements at every level
3. Give each element a coherent purpose
4. Use encapsulation techniques to not over-share internal details

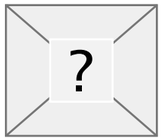




Completeness

- Not all levels / not all elements will need the same level of details
- This does not need to be comprehensive
  - More complex areas, will probably have more comprehensive design models
  - Areas with more engineers will need more design models

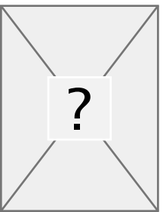


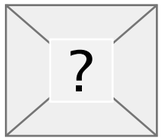


Understandable Systems

- Hierarchy is usually introduced to make systems understandable

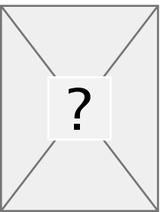
- This is why object oriented languages have remained so popular, as they lend themselves to creating hierarchical systems with ease

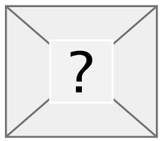




Non-hierarchical systems

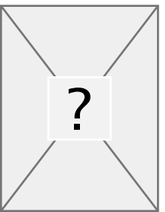
- The book gives a hardware example
- Practical software examples:
  - I can’t think of any
  - I have seen non-hierarchical, flat systems
    - Their code was a mess until we refactored it to be hierarchical

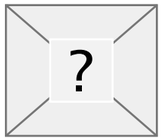




Top-down design

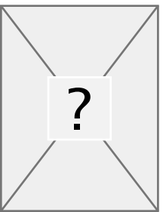
- A hierarchy doesn't imply that you always design the system from the top down
- Specific sub-components near the bottom may be the most risky / have the long timelines
  - It is reasonable to design those first

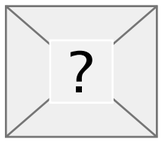




Decomposition strategies

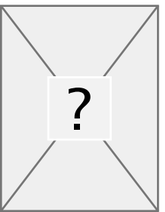
- Functionality
- Archetypes
- Architectural style
- Attribute Driven Design
- Ports
- Orthogonal abstraction
- Jigsaw puzzle

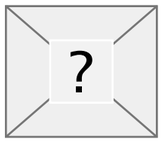




Functionality

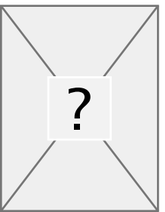
- Group your system by related functionality and decompose it along those lines
- Example
  - Registration
  - Administration
  - Main user functionality

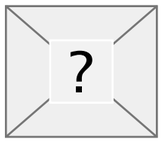




Archetypes

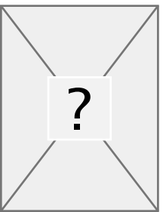
- Archetypes from the domain
    - Contact application: applying the vcard format

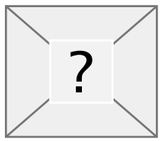




Architectural style

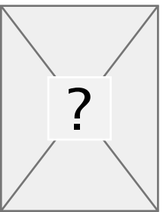
- If you select a 3 tier architecture, then you will decompose your system that way
- If you select a peer-to-peer architecture, etc…

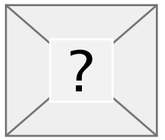




Attribute Driven Design

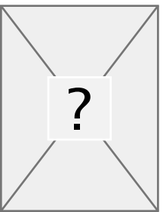
- Comes from the Software Engineering Institute
- Decide which quality attributes are the most important for a component
  - Choose a design that is suited for achieving those attributes

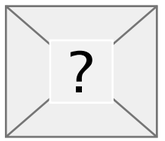




Ports

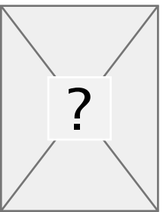
- “Ports” is a loose definition for communication points with other components
- The port represents a distinct grouping of functionality or responsibility
- To me: This is closely related to / basically the same thing as, functional decomposition

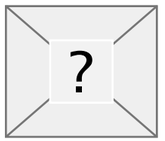




Orthogonal Abstraction

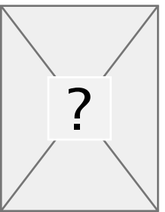
- Restate the problem in a different domain
- example: work order system into a directed graph, dependency problem
- Some domains have well known patterns
- To me, this is like reducing one NP-complete problem to another
    - Just thinking of something differently

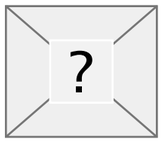




Jigsaw puzzle

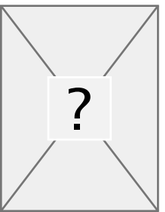
- Putting together a design from some already existing pieces
- More like a systems integration project
  - Common in IT projects

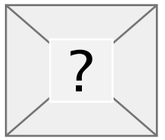




Effective Encapsulation

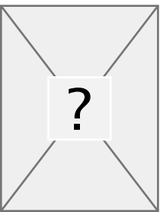
- If decomposition is splitting a solution into parts
- Encapsulation is hiding implementation the details in those parts
- This is a standard principle of modern, object-oriented software design
  - Most of the time, we care about what a system does, and less about how that is accomplished
  - There are times where it is appropriate to be worried about the “how”

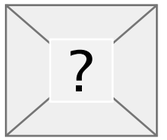




Complexity Reduction

- Encapsulation reduces complexity when reasoning are the whole of, or large parts of a system
- This aids in your design process as the system scales





Encapsulation Failures

- `