Analysis Heuristics

Dr. M.E. Fayad, Professor
Computer Engineering Department, Room #283I
College of Engineering
San José State University
One Washington Square
San José, CA 95192-0180
http://www.engr.sjsu.edu/~fayad
Lesson 5: Analysis Heuristics
Lesson Objectives

- Overview of previous lectures.
- Understand the analysis heuristics
  - Go Beyond the Problem Domain
  - Speculate About Likely Changes
  - Separate General Functionality from Specific Policy
  - Object should have Cohesive Interfaces
  - Objects Should Be Intelligent Agents
  - Objects Should Export Services
  - Avoid “Object Machismo”
### Overview: Life Cycle

<table>
<thead>
<tr>
<th>RA</th>
<th>What</th>
<th>What is the customer really wants?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>How</td>
<td>How – the best solution</td>
</tr>
<tr>
<td>Code</td>
<td>Build</td>
<td>How do we construct (implement) the system</td>
</tr>
<tr>
<td>Test</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Are customer requirements testable?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Does the how logically follow from what?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Does the built system do what it is suppose to do?</td>
</tr>
<tr>
<td>Deploy</td>
<td>Use</td>
<td>How do we enhance &amp;/or repair the built system?</td>
</tr>
</tbody>
</table>
Overview: Analysis vs. Design

- Analysis
  - What is the problem?
  - Problem Space
    - Mostly “one” Problem
  
- Design
  - How to solve the problem?
  - Solution Space
    - Many Solutions
Overview: Process Properties

- Practical
- Concrete Action
- Can be measured
- Repeatable
- Tailorable
- Must be documented
- Hierarchical
- Specify who, what, when, how and ignore the why?
Overview: Model Properties

- Simple
- Complete (most likely to be correct)
- Stable to technological changes
- Testable
- Easy to understand
- Visual or graphical
Heuristic #1: Go Beyond the Problem Domain

- A system structure is based on the "Real World", "locks in" today’s problem domain relationships.
- This makes it difficult to adapt the system to future requirements.
- Look for relationships and generalizations that transcend the current problem domain
  - Ask "What is it?"
Heuristic #1: Go Beyond the Problem Domain

- Build these into the analysis model
- Developing an adaptable architecture does not happen just because you are using OOD and/or C++ (any more than extensibility or reuse occur automatically)

Generalize Early & Generalize Vigorously
Heuristic #2: Speculate About Likely Changes

- The “Real World” is the best model of today but it only hints at what tomorrow will bring
- Basic for speculation
  - Changing user requirements
  - Changing customer base
  - Competitive products
  - Changing technology
Heuristic #2: Speculate About Likely Changes

- Build the analysis model so it can adapt to these likely changes
- You do not have to be 100% correct.
- Developing an adaptable architecture does not happen just because you are using OOD and/or C++ (any more than extensibility or reuse occur automatically) – Exs: Hooks, HotSpots, Design Patterns
Heuristic #3: Separate General Functionality from Specific Policy

- Place general functionality in entity objects (class)
  - Entity object (class) will now be more reusable.
- Place specific policy in control objects (active class)
  - Policy is localized so that it is easier to introduce changes to this policy
Heuristic #3: Separate General Functionality from Specific Policy

- Conservation of Policy
  - There is no way to eliminate or ignore all policy
  - The policy will be in the delivered system
  - All we can do is structure the policy so that it is easy to adapt and change it.

Mechanism Rich & Policy Free
Heuristic #3: Illustrated Example – The Problem
Heuristic #3: Illustrated Example – The Solution

State Tax

Federal Tax

Forms

Calculator

File System

MRPR

MRPR

MRP+

MRPF

MRPF

© M.E. Fayad 1996-2006

SJSU – CmpE     M.E. Fayad     L5-S15     Analysis Heuristics
Heuristic #4: Objects Should Have Cohesive Interfaces

- In the “Real World”, a remote controller for home electronics may have 50 buttons for controlling your TV, VCR, Stereo, and others.
- Modeling this controller as a single, real-life object will not be adaptable to future changes.
Heuristic #4: Objects Should Have Cohesive Interfaces

- It may be better to:
  - First model each different set of operations as a separate object with a strongly cohesive interface.
  - Model the combined functionality as a separate object that uses other objects.

- Result is more adaptable and reusable

Do One Thing
Heuristic #5: Objects Should Intelligent Agents

Intelligent (Responsible) Objects

- Incorporate important knowledge
- Incorporate knowledge that is difficult to produce, find, or replicate
- Know how to synthesize knowledge
Heuristic #5: Objects Should Intelligent Agents

Agents are capable of (limited) autonomous behavior

- Know that they are supposed to do, and they do.
- Work best with limited supervision.
- Adapt to their environment
- Know how to delegate work to other objects
Heuristic #5: Objects Should Intelligent Agents

Agents are capable of (limited) autonomous behavior

- Know how to find objects to which they can delegate work
- Have the information they need or know where to get the information or know where to get information on getting information or can interpret information given to them.
- Are highly adaptable, extensible, and reusable
- Are expensive to design and build
Heuristic #6: Objects Should Export Services

- Objects that only export attributes or data must be manipulated by clients. (aka. “dead data”)
- Objects that only export basic operations require clients to direct and supervise all activity (aka. “stupid objects”)
- Objects that export services make life easier for their clients:
  - Server selects the best way to perform the service
  - Server finds and manages the resources
Heuristic #6: Objects Should Export Services

- Services should define “What” not “How”
  - Can “Drive to Work” be replaced by “Get to Work”
  - Enhances extensibility
  - Enhances reusability
  - Distributes intelligence

Move Complexity From the Clients to the Servers
Heuristic #6: Stack Example

Stack Implementations

Type Stack

- push()
- pop()
- length()
- empty()
- full()

Stack Interfaces
Heuristic #7: Avoid “Object Machismo”

Object machismo

– Equating the value of an object with how big and/or complex it is (e.g., Lines of Code, # of methods, or complexity of algorithms it uses).

– Macho objects tend to be central controllers that are difficult to design, difficult to understand, have to much policy, are hard to extend, and low reuse potential.
Heuristic #7: Avoid “Object Machismo”

The value of an object is based on many factors:

- Does it do something useful?
- Does it have a simple and clean interface?
- Does it have well-specific behavior?
- Does it model an essential quality of the system?
- Can it be composed with other objects to perform more complex tasks?
Heuristic #7: Avoid “Object Machismo”

- The value of an object is also based on other objects:
  - An object perfectly suited for one model may be totally wrong in another model
  - The object must be placed in context to see whether or not it is a good and valuable object.

A Valuable Object Works and Plays Well with Others.
Discussion Questions

• Explain the following statements:
  1. Objects should be intelligent agents
  2. Mechanism rich and policy free
  3. A valuable object works and plays well with others
  4. Analysis model should not be too elaborate or too formal

• Explain how to build an analysis model
• Explain how do you make the analysis model more adaptable
• Essay Topic: Analysis Guidelines