Constructing Object oriented programs
Which Design is Better?
A Talk in CSE Seminar Series of 2007-2008
March 12, 2008, 2-3 pm

Rushikesh K Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Outline

1. Object Oriented Designs
2. Design Before Implementation?
3. A Methodology Problem
4. Actual Results on a Local Software
5. Further Design Improvements and Tool Support
Outline

1. Object Oriented Designs
2. Design Before Implementation?
3. A Methodology Problem
4. Actual Results on a Local Software
5. Further Design Improvements and Tool Support
Object Oriented Designs

Decomposition of the abstraction space
- Classes and Objects

Weave them through various relationships

The Goal?
- Core Functionality and
- Design Goodness Criteria
The Development Paradigm

Design, Implementation

Model Driven Design and Extreme Programming
Outline

1. Object Oriented Designs
2. Design Before Implementation?
3. A Methodology Problem
4. Actual Results on a Local Software
5. Further Design Improvements and Tool Support
A designer may create many alternatives

What kinds of flaws may exist in designs?

let’s look at some examples..
A Dynamic Functionality Enhancement Problem
Design Feature-1

Enhancers (Decorators) Enhance the Core Functionality

The core functionality is not removed
Dynamic Pluggability – One should be able to attach and detach decorators dynamically
Design Feature-3

Caller code independent of the difference that is being made to the quality of service

The calling application sees the same interface for decorator and end-server
No. of decorators: 0..n
Proposal 1

client ➔ Server

dec1 ➔ dec2
Pros and Cons

- Okay for delegation based inheritance model (with a minor modification)
- Does not work with class-based inheritance
- Chaining of independent decorators at object level not achieved
- Decorators are bulky, redundancy- multiple copy problem
Proposal 2

```
client

Server

Abstract

dec

decl

dec2
```
Pros and Cons

- Chaining at object level
- interface compatibility
- Chaining does not scale up (cannot add more decorators)
- Decorators are bulky, redundancy- multiple copy problem
Proposal 3

![Diagram]

- Server
- dec
- client
- dec1
- dec2
Pros and Cons

- Confusion between caller and callee abstractions
- Member function invocations that go in project the interface, and not the invocations that come out
Proposal 4
Pros and Cons

- Generality among decorators captured
- Only one decorator at a time
- Server part of decorator
Proposal 5

```
server
  client
  \_ dec
    \_ decNull
    \_ dec1
    \_ dec2
```
Pros and Cons

- Generality among decorators captured
- Only one decorator at a time
- Server part of decorator
- Default Null Decorator—Is it necessary?
Proposal 6
Pros and Cons

- Server disconnected from decorator, caller:
- A Relation from server to decorator and not vice versa
- Dec1 part of Dec2
Proposal 7
Pro and Cons

- Comes close - decorators, servers are independent
- Only one decorator at a time, no chaining
- Commonality between decorators and server not captured
Proposal 8

Object Oriented Designs
Design Before Implementation?
A Methodology Problem
Actual Results on a Local Software
Further Design Improvements and Tool Support
Pros and Cons

- Recursive Chaining due recursion at generic level
- server is aware of decoration - Is outgoing chaining from the server needed?
Proposal 9
Pros and Cons

- Decorators are chained
- Only one decorator type benefits from the chaining
- Server has no outgoing links
- But decorators cannot connect to server
Proposal 10
Pros and Cons

- Commonality captured beautifully
- Chaining is static (type specific) since it’s class-class chaining
Proposal 11
Commonality among decorators captured
Decorators are not servers themselves
The designer explicitly captured 3 possibilities-
1-s, 2-1-s, 3-2-1-s
Too many links, static chaining
Pros and Cons

- Decorators chained
- Only one type of chained decorator
- Commonality between decorators and server not captured
Proposal 13
Pros and Cons

- Top level decorator knows of the rest
- Front decorator is also server, the rest are not
- Rest of the decorators cannot be used independently, two interfaces
Proposal 14
Pros and Cons

- Decorators chained
- Every decorator is also a server
- Only one type of decorator benefits from chaining
- Cannot add subclasses- with chaining between instances of existing classes and the newly added subclass, the design will get messed up
Proposal 15

Object Oriented Designs
Design Before Implementation?
A Methodology Problem
Actual Results on a Local Software
Further Design Improvements and Tool Support
Pros and Cons

- Decorators chained generically
- Decorators cannot reach the server
- Commonality between decorators and the server not captured
Proposal 16
Pros and Cons

- Looks more like an object model than class model
- Front decorator knows of the rest of the decorators and the server
- Cannot add a new decorator without modifying the existing front class’s state and code
Proposal 17
Pros and Cons

- Recursive chaining captured through upward links + inheritance
- All decorators are servers
- Genericity in chaining not captured
Object Oriented Designs
Design Before Implementation?
A Methodology Problem
Actual Results on a Local Software
Further Design Improvements and Tool Support

Proposal 18

Diagram: Diagram showing the relationship between client, server, abstract server, server, decl1, and decl2.
Pros and Cons

- Quite close!
- Decorators chained at generic level
- The fact that each decorator may have at most one downlink is not captured generically
Proposal 19
Pros and Cons

- Decorators chained at generic level
- The fact that each decorator may have at most one downlink is captured generically through an intermediate abstraction
Structural Properties of the Last Solution

- Server instance is the end object in a chain
- Decorators and Servers share the same abstraction
- New decorator benefits from chaining captured at a generic level
- Decorator chaining configuration is not statically committed
Front Deletion: still a problem

- Client needs to be contacted for deleting or replacing the front decorator
- Having a default front is an overhead when decorators are not used
- To solve this problem: What’s needed is location transparency
Outline

1. Object Oriented Designs
2. Design Before Implementation?
3. A Methodology Problem
4. Actual Results on a Local Software
5. Further Design Improvements and Tool Support
Summary

*Design before Implementation* requires a careful analysis of the design, if the design should sail straight into implementation.

- Capture the properties that are desirable
- Do not capture the properties that are undesirable

An extreme approach creates more problems as the system evolves, especially when refactorings are not applied continuously!
## Impact of Design Improvement

<table>
<thead>
<tr>
<th>class name</th>
<th>NOM before</th>
<th>LOC before</th>
<th>NOM after</th>
<th>LOC after</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpApplicant</td>
<td>50</td>
<td>1776</td>
<td>50+11</td>
<td>1118</td>
</tr>
<tr>
<td>ConnectDatabase</td>
<td>10</td>
<td>265</td>
<td>10</td>
<td>220</td>
</tr>
<tr>
<td>MtechApplicationInit</td>
<td>33</td>
<td>565</td>
<td>33+3</td>
<td>450</td>
</tr>
<tr>
<td>MtechApplication</td>
<td>8</td>
<td>339</td>
<td>8</td>
<td>300</td>
</tr>
<tr>
<td>MtechMailer</td>
<td>3</td>
<td>60</td>
<td>3</td>
<td>60</td>
</tr>
<tr>
<td>StatusMailer</td>
<td>1</td>
<td>35</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>ValidationChoice</td>
<td>7</td>
<td>131</td>
<td>7+4</td>
<td>114</td>
</tr>
<tr>
<td>ImpData</td>
<td>0</td>
<td>35</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Constants</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Significant improvement in LOC mainly by method extraction and no major design changes were applied.
Outline

1. Object Oriented Designs
2. Design Before Implementation?
3. A Methodology Problem
4. Actual Results on a Local Software
5. Further Design Improvements and Tool Support
Cohesion and Coupling

Apply Structural Metrics to detect problem areas

There are Many Metrics and they need structural information
— An intermediate representation for OO programs– metric friendly

Metrics are mainly class-based– they donot pinpoint problem methods
— Microscopic Metrics
A Compact Model for Measuring Designs and Code
Limitations of Object orientation

- Advanced Separation of Concerns (beyond what can be expressed in OOP)
- Code is redundant, but cannot be modularized
- Capture such code through aspects and weave them with bases
- Need to Measure the goodness of Aspectized Code → Metrics for AOP