Final Exam:
Take-home versus In-class

- Take-home
  - No rush to complete in 1 hour at 9:40am
  - More realistic (but same number of) problems
    - Still bounded complexity
  - Different atmosphere from the midterm, tests different skills
  - Access to other (online) resources in responding to questions
  - One more day in the class for covering actual subject content
- In-class
  - Like a second midterm, but higher stakes (20% of grade)
  - On the last day of class
  - Course would end with an exam, rather than with a pleasant discussion and retrospective
- Hybrid version?
  - In-class for 2 hours (Thu @ 8:40am-10:40am or 9:40am-11:40am)

Final Release Deliverables

- **Installation packages**
  - Including all of the items below
- **Application sources and binaries**
  - Separate distributions (installation packages) for customers and developers
  - One-step build – from compiling all sources to creating installation packages
- **User & technical documentation (separate)**
  - User doc: What does my mom need to know (and do) in order to run your product?
  - Technical doc: What does a new dev't team need to know to work on version 2?
- **Release notes**
  - Known remaining issues with associated severities & priorities
    - Include a link to your bug tracking system's tasks/tickets reflecting those issues
  - Location of your current code repository
  - Instructions on running the installer and your app are now moved to the user doc.
- **Latest test plan**
  - Reflecting the test cases that were verified as well as those that were left out
- **Automated tests (unit tests and acceptance/full system tests)**
  - Information on test coverage would be a very welcome (but optional) addition.
- **Up-to-date schedule**
  - Items that have been accomplished (of those that were planned/promised)
  - Features (of those initially planned) that are now pushed to version 2 or abandoned
    - How much would each such feature cost (in terms of development effort)?

Questions to consider: Who is your audience – customers or developers? What do they expect from this release? What defines success for them?

Lecture 16: Design Patterns, Cont

Viewing problems and solutions in context

Design Patterns Defined

> ***"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"***
>
> **– Alexander, et al., A Pattern Language. Oxford University Press, 1977**

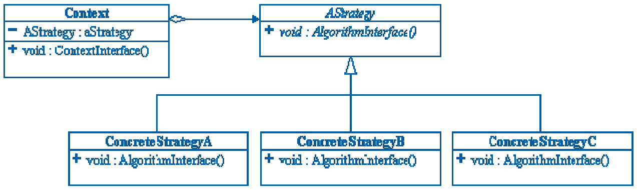
Pattern Principles

+ Encapsulate variance
+ Abstract the invariant
+ Favor composition

Strategy

+ **Intent:** Encapsulate each of a family of behaviors such that the use of the behaviors varies independently of the client.
+ **Motivation:** Remember the animals…

Strategy: Structure



Context
- AStrategy : aStrategy
+ void : ContextInterface()

AStrategy
+ *void : AlgorithmInterface()*

ConcreteStrategyA
+ void : AlgorithmInterface()

ConcreteStrategyB
+ void : AlgorithmInterface()

ConcreteStrategyC
+ void : AlgorithmInterface()

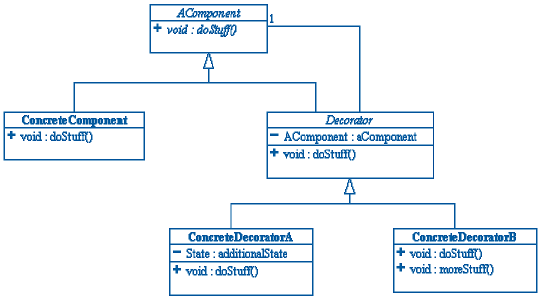
A New Pattern

+ **Intent:** Attach additional responsibilities to an object dynamically or in response to changing requirements.
+ **Motivation:** What if we need to add additional processing steps to one of our algorithms…

Student Submission

A New Pattern, Continued

Decorator: Structure



AComponent
+ *void : doStuff()*

1

ConcreteComponent
+ void : doStuff()

Decorator
- AComponent : aComponent
+ void : doStuff()

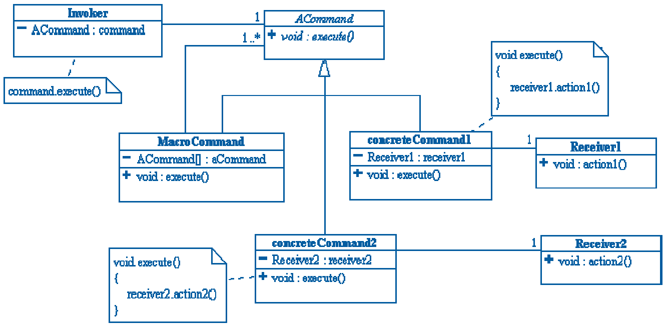
ConcreteDecoratorA
- State : additionalState
+ void : doStuff()

ConcreteDecoratorB
+ void : doStuff()
+ void : moreStuff()

Command

+ **Intent:** Encapsulate a request or action as an object.
+ **Motivation:** Sometimes we need to issue commands without knowledge of the specifics of any command – a menu is a good example.

Command: Structure



Invoker
- ACommand : command

```
command.execute()
```

1

1..*

ACommand
+ *void : execute()*

```
void execute()
{
    receiver1.action1()
}
```

MacroCommand
- ACommand[] : aCommand
+ void : execute()

concreteCommand1
- Receiver1 : receiver1
+ void : execute()

1

Receiver1
+ void : action1()

```
void execute()
{
    receiver2.action2()
}
```

concreteCommand2
- Receiver2 : receiver2
+ void : execute()

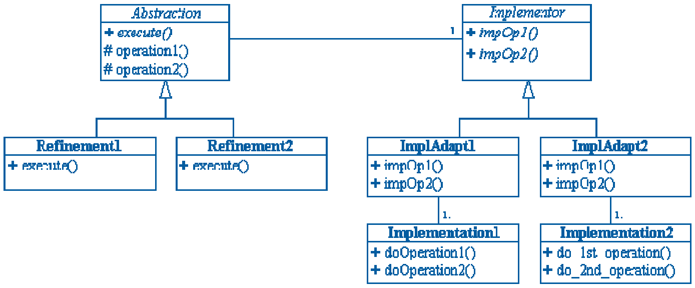
1

Receiver2
+ void : action2()

Bridge and Adapter

+ **Intent:** Decouple a set of implementations from the set of clients using them.
+ **Motivation:** Perhaps we want to enable different toolkits to perform the same operations…

Bridge and Adapter: Structure



Resources

+ Hillside.net (http://hillside.net/patterns/)
+ Patterns and Frameworks (http://www.cs.wustl.edu/~schmidt/patterns.html)
+ Design Patterns: Elements of Reusable Object-Oriented Software. Gamma et al. Addison-Wesley, Boston, 1995.
+ Design Patterns Explained: A New Perspective on Object-Oriented Design, 2nd Ed. A. Shalloway and J. Trott. Addison-Wesley, Boston, 2004 .

Lecture 19: Software Quality

Outline

Today:

- Quality – a look back at history
- "Good enough" quality
- What is (software) quality?

Next time:

- How do we measure quality?
- How do we improve software quality?

References

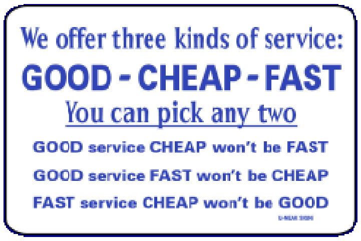
- *Good Enough Quality: Beyond the Buzzword*, by James Bach
  - http://www.satisfice.com/articles.shtml
- *Professional Software Development*, by Steve McConnell
- *Agile Software Development with Scrum*, by Ken Schwaber and Mike Beedle

Food for Thought: Quality in Different Contexts

- The software 'Gold Rush' fever periods
  - Goal: being first-to-market in an unclaimed segment
  - Typical environment: two guys in a garage
  - High-risk projects, potentially high pay-off
  - Code-and-fix development, very informal processes
  - Customers are tech savvy, willing to forgive bugs
- The in-between (post-'Gold Rush') periods
  - Goal: sustained, productive competition with others
  - Typical environment: larger teams, formal processes
  - Lower-risk, likely lower but more predictable pay-off
  - Careful, quality-driven development with an emphasis on quality (reliability, interoperability, usability, etc.)
  - Different customer base: demands reliability

Back to Basics: The Goal of Building Software

- To deliver a product that satisfies the customer(s)
  - on time
  - on budget
  - with good quality
- But wait…



We offer three kinds of service:

GOOD - CHEAP - FAST

You can pick any two

GOOD service CHEAP won't be FAST

GOOD service FAST won't be CHEAP

FAST service CHEAP won't be GOOD

The Quality Question

How do we ensure good quality for software?

"Good Enough" Quality

(James Bach, http://www.satisfice.com/articles.shtml)

- At some point, one needs to stop and decide it is all good enough to ship. Under what conditions?

Criteria for "good enough" quality:

1. There are clear benefits (of the software).
2. There are no critical problems.
3. Overall, the benefits outweigh the problems.
4. In the present situation and all things considered, further development would be more harmful than helpful.

Question: Is your product good enough *now* ?

"Good Enough" Quality (cont.)

Important questions to consider:

- Good enough *for whom*?
  - You? Your team? The customer?
- Good enough *for what*?
  - A demo? A beta release? Selling it? Capturing market share?
- Have you agreed on …:
  - team standards for acceptable quality?
  - what would constitute success for your team in the end?
    - These are some of the team conversations we did earlier.

Take a Step Back: What is Quality?

- Quality is in the eyes of the beholder (customer).
  - First, define what quality means (for the customer!).
  - You and the customer must agree on the expected level of quality.
- What constitutes good quality in one situation may not be considered good quality in another.
  - E.g.: in a toy project vs. in a safety-critical system
- A *contract* must have at least the following components:
  - **Who** promises to do
  - **What**
  - for **Whom**
  - by **When**
  - with what **Quality Criteria**/Standards, and
  - with what **Notification Mechanism** upon completion

Student Submission

Components of Quality

- Quality comprises (but is not limited to):
  - Requirements quality
  - Design quality
  - Code quality
  - Test quality
  - Documentation quality

- Given limited resources, which of these do you consider more important to pay attention to? Why?