Software Tools & Methods
Class 6

Lecturer: Greg Bolcer, greg@bolcer.org

Summer Session 2009

ELH 110 9am-11:50am

Overview

- Last Class
  - Midterm: 38 B+/A-
- This Class
  - HW4/Lab
  - Midterm review
  - More design patterns
  - Agile, XP, Scrum revisited
- Next Class
  - More design patterns
  - Use cases

HW4 & Lab

- Due date extended to 7/17 for full credit
- Problems with Lab
  - Check your shell: tcsh vs. bash
  - Check subversion module number: subversion/1.5.5
  - Installing the correct version of Subclipse
    - Subclipse 1.4.x includes and requires Subversion 1.5.x client features and working copy format
    - Update URL: http://subclipse.tigris.org/update_1.4.x
  - Follow instructions very carefully both in HW handout an on instruction pages

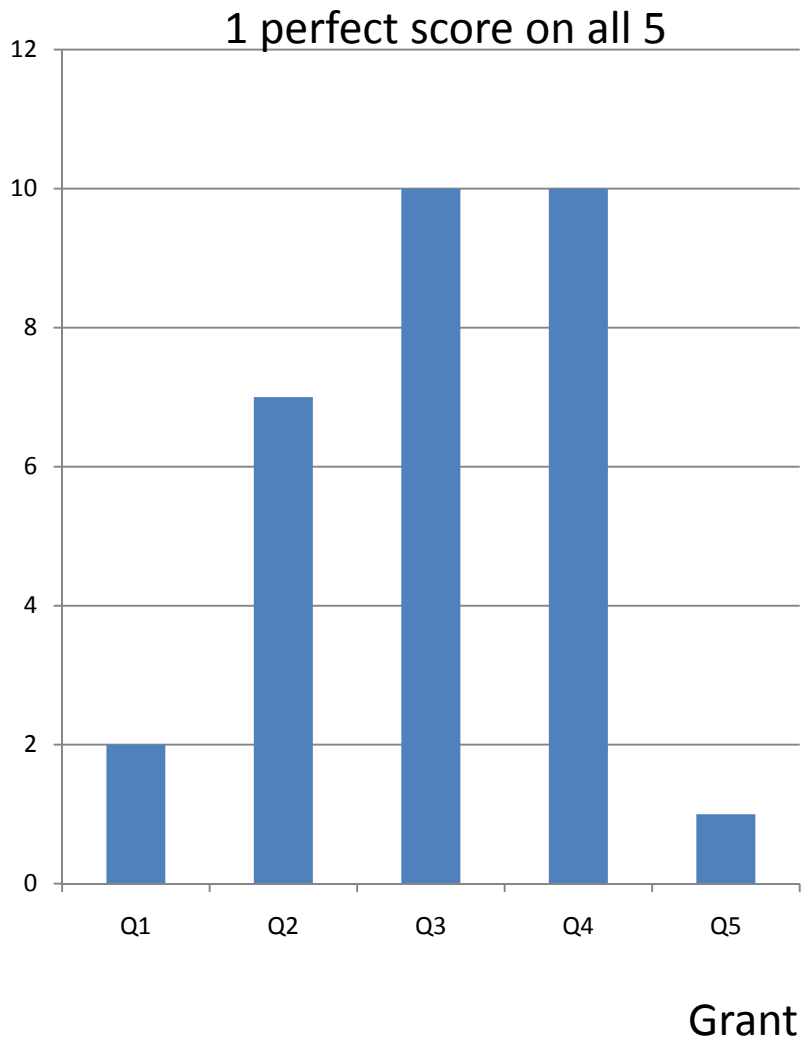
Software Tools & Methods

MIDTERM

Study Review

- Silver Bullet Accidental vs Essential difficulties
- Mythical Man Month
- Orders of Ignorance in Software Process
- UML Generalization, Aggregation, Dependency, Composition, Realization; Types of UML diagrams
- Coding standards
- Version Control Optimistic/Pessimistic, version numbers
- Tool chains—be able to name a couple of tools and describe what they do
- Agile, XP, Rational Unified Process, RUP phases and activities
- UML Class Diagrams, public, protected, private; Classes, Attributes, Operations

Multiple Choice



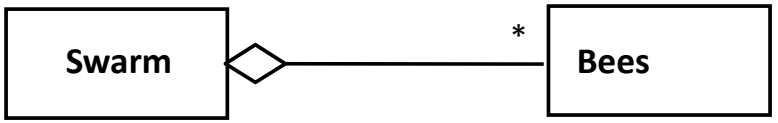
- A software method for unit testing
- A software method for reverse engineering
- A software method for formatting code
- *A software method for Spiral Model*
- A software method for evolving requirements

Multiple Choice

- Software process models help reduce which Order of Ignorance
- Lack of process, i.e. you don't know how to go about discovering the knowledge

Aggregations are special associations that represent 'part-whole' relationships.

- The 'whole' side is often called the *assembly* or the *aggregate*
- This symbol is a shorthand notation association named **isPartOf**



Multiple Choice

- Reading code from top to bottom is NOT how to do code reading
- Once you elminated this, only one answer left

- Subversion
  - Key to this question is understand optimistic version control
  - '*' represented changes to the working copy
- Draw circle with repository, cross out and put new number, update user's copies

Short Answer

- 2.1 (8 points)
  - 1 perfect score
- Perfect score needed to have
  - Mention of adding more people making project later
  - Mention of components as easily separable or no communication
- Poor management, expertise, poor staffing, etc. go against what Brooks was trying to say, but no penalty points taken off

Naraghi

Short Answer

- 2.2 (20 points)
  - Highest score was 19
  - 10 Points for correct naming
  - 1 point each for correct mapping
  - Partial credit for similar phase names
- Inception
  - Defining Scope of System
  - Life-cycle Objectives milestone
  - Outlining a candidate architecture

Fuentes

Short Answer

- Elaboration
  - Life-cycle Architecture milestone
  - Capturing a majority of functional requirements
  - Addressing significant risks on an ongoing basis
- Construction
  - Building a system that operates successfully
  - Initial Operational Capacity milestone
- Transition
  - Rolling out fully functional system to customers
  - Product release milestone

Long Answer

- Largest point deductions
  - No use of extends and implements keywords
  - No data structure to show one to many relationship between credit card report and credit card transaction
  - One point here or there for declarations in wrong place, wrong attribute
- Make sure all classes, all attributes, all methods, all declarations

Software Tools & Methods

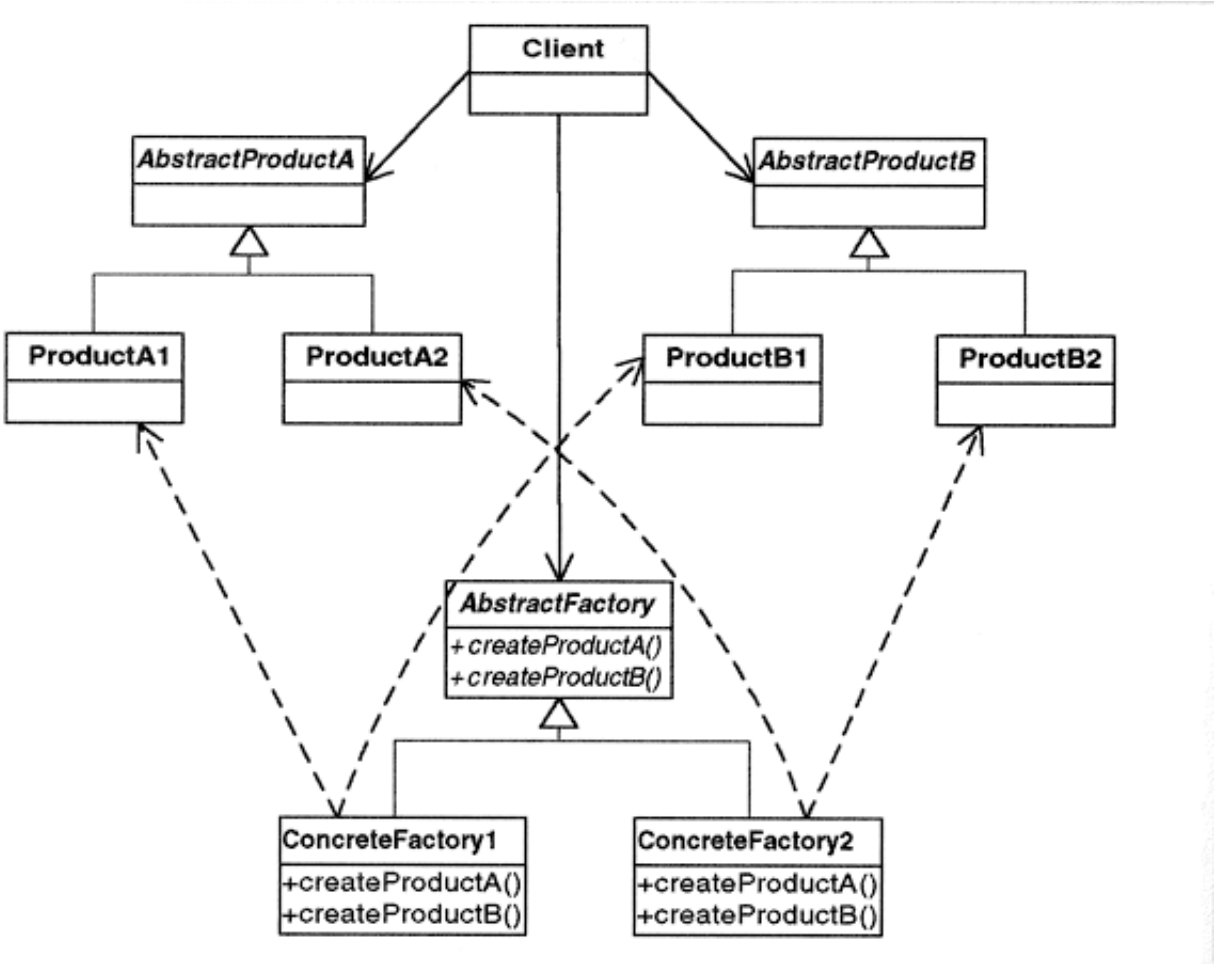
DESIGN PATTERNS CONTINUED

Abstract Factory Pattern

- Context
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Also known as
  - Kit
- Problem
  - Who should be responsible for creating objects when there are special considerations, such as complex creation logic, a desire to separate the creation responsibilities for better cohesion, and so forth?
- Solution
  - Create and object that handles the creation of the objects

Abstract Factory Pattern

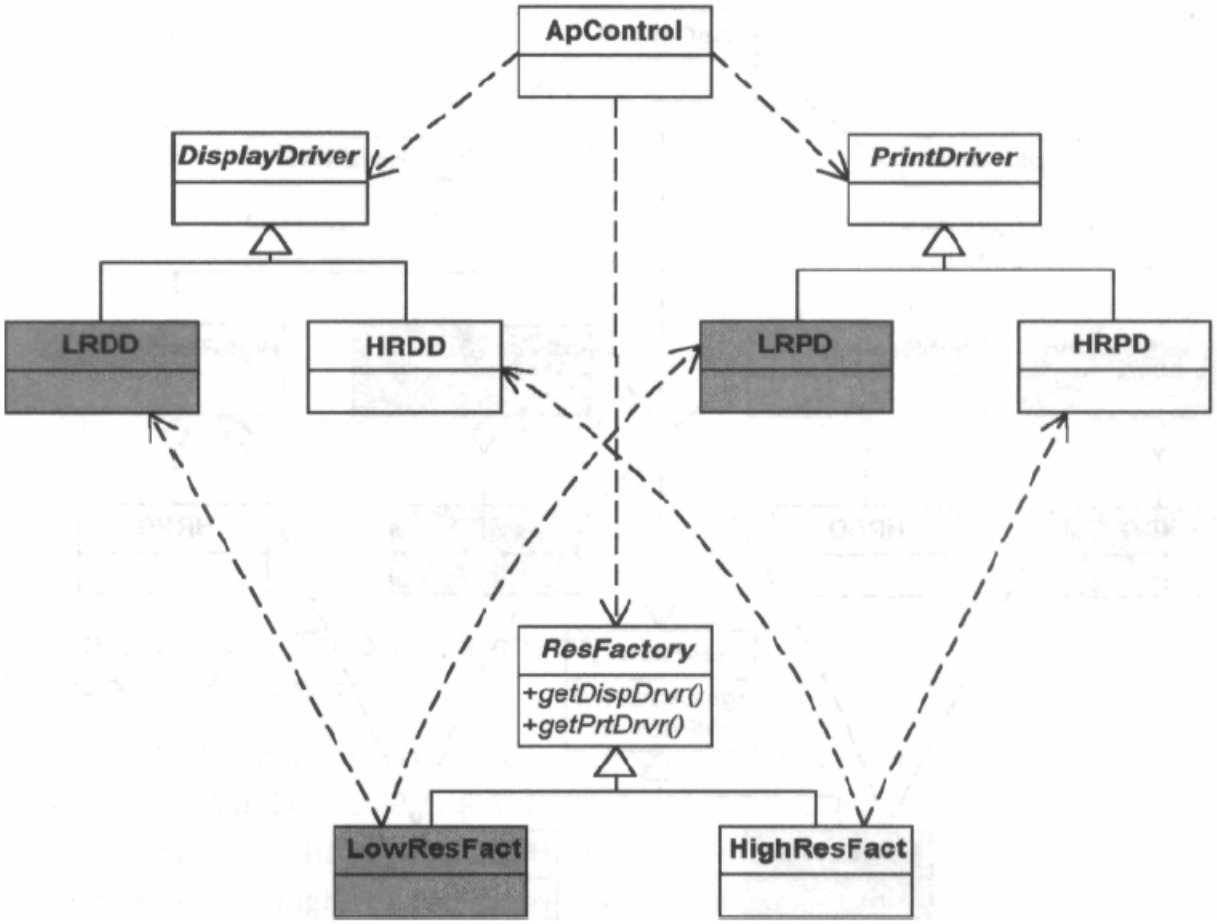
- Structure



Example

- Different Drivers for Different Platforms

For Driver	Low-Capacity Machine	High-Capacity Machine
Display	Low-Resolution Display Driver LRDD	High-Resolution Display Driver HRDD
Print	Low-Resolution Print Driver LRPD	High-Resolution Print Driver HRPD


ApControl
DisplayDriver
PrintDriver
LRDD
HRDD
LRPD
HRPD
ResFactory
+getDispDrvr()
+getPrtDrvr()
LowResFact
HighResFact

Abstract Factory Pattern

- Participants
  - Abstract Factory defines the interface for how to create each member of the family of objects. Typically, each family is created by having its own unique Concrete Factory.
  - The Concrete Factory to use is usually specified in a file (not implemented in program logic or global variable)

Abstract Factory Pattern

- Consequences
  - The pattern isolates the rules of which object to use from the logic of how to use these objects.
  - Exchanging kits of objects becomes easier.
  - Promotes consistency among products.
  - Supporting new kinds of products is harder.

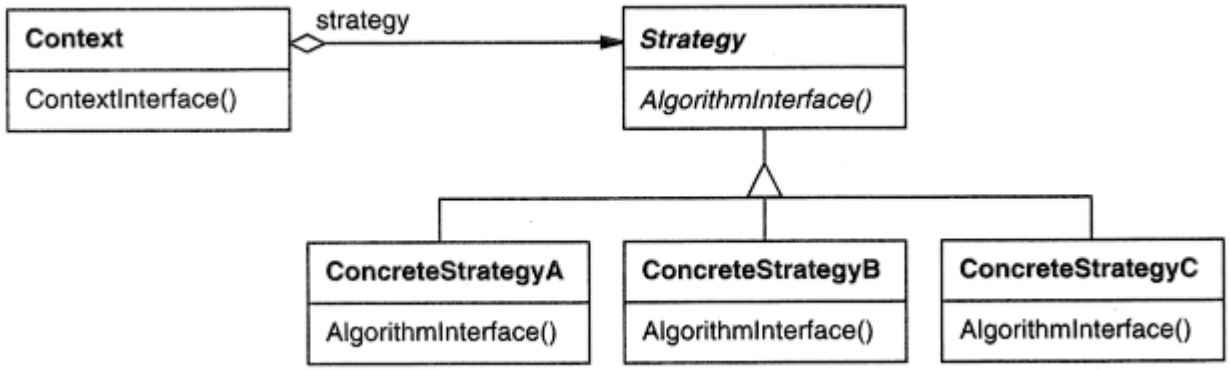
- Implementation
  - Define an abstract class that specifies which objects are to be made. Then implement one concrete class per family. Tables or files can also be used to accomplish the same thing.

Strategy Design Pattern

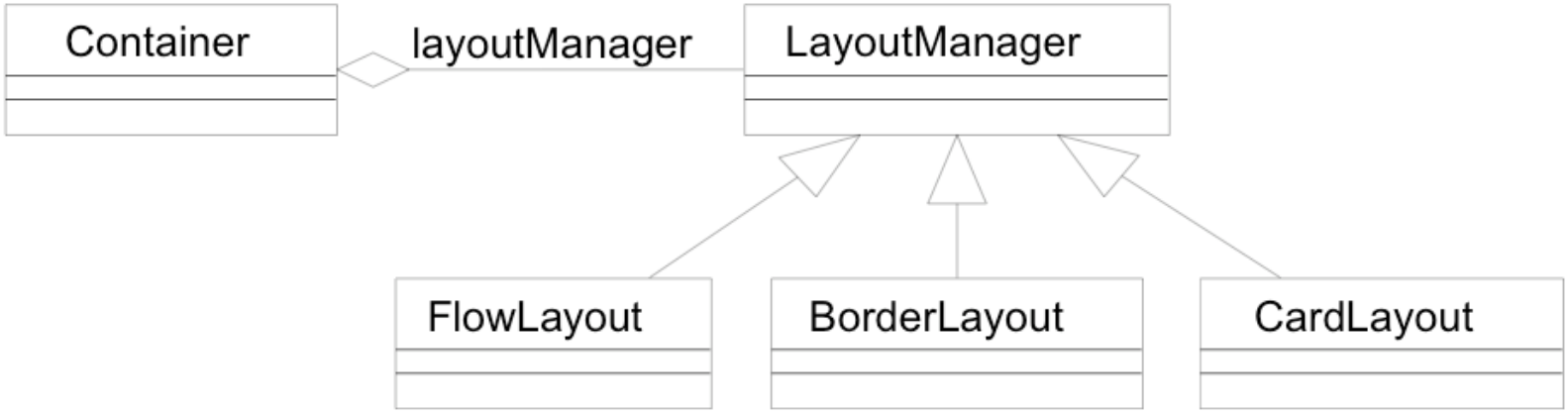
- Context
  - Define a family of algorithms, so they are interchangeable.
- Also Known As
  - Policy
- Problem
  - How to design for varying, but related algorithms or policies? How to design for the ability to change the algorithms or policies?
- Solution
  - Define each algorithm/policy/strategy in a separate class with a common interface

Strategy Design Pattern

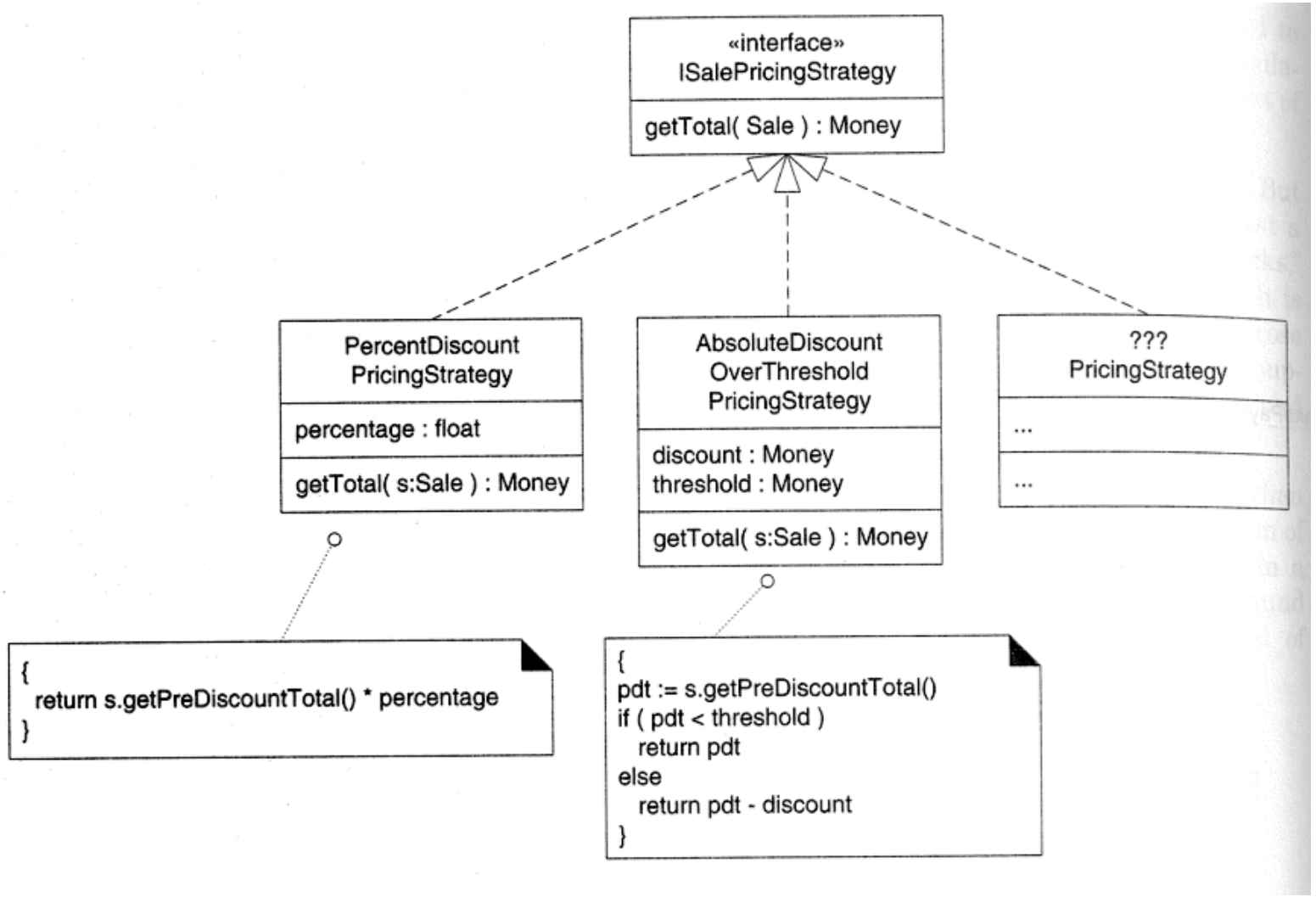
- Structure



Example


Container
layoutManager
LayoutManager
FlowLayout
BorderLayout
CardLayout

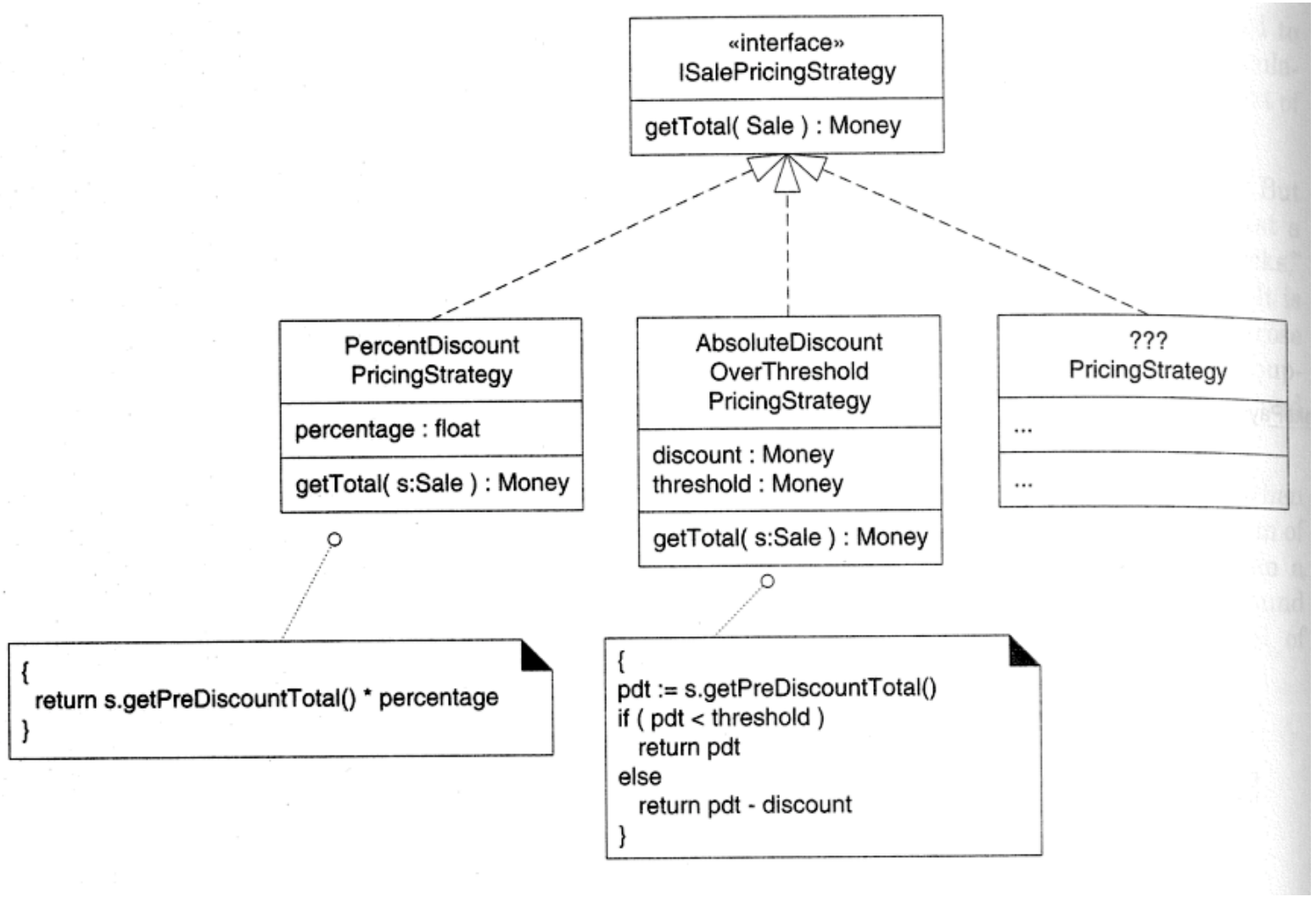
Example


«interface»
ISalePricingStrategy
getTotal(Sale) : Money
PercentDiscount
PricingStrategy
percentage : float
getTotal(s:Sale) : Money
AbsoluteDiscount
OverThreshold
PricingStrategy
discount : Money
threshold : Money
getTotal(s:Sale) : Money
???
PricingStrategy
...
...
{
return s.getPreDiscountTotal() * percentage
}
{
pdt := s.getPreDiscountTotal()
if (pdt < threshold)
return pdt
else
return pdt - discount
}

Strategy Design Pattern

- Participants
  - Strategy interface, concrete Strategy, and Context/client
- Consequences
  - Provides an alternative to subclassing the Context class to get a variety of algorithms or behaviors
  - Eliminates large conditional statements
  - Provides a choice of implementations for the same behavior
  - Increases the number of objects
  - All algorithms must use the same Strategy interface
- Implementation
  - Can use an Abstract Factory to create a Strategy

Example

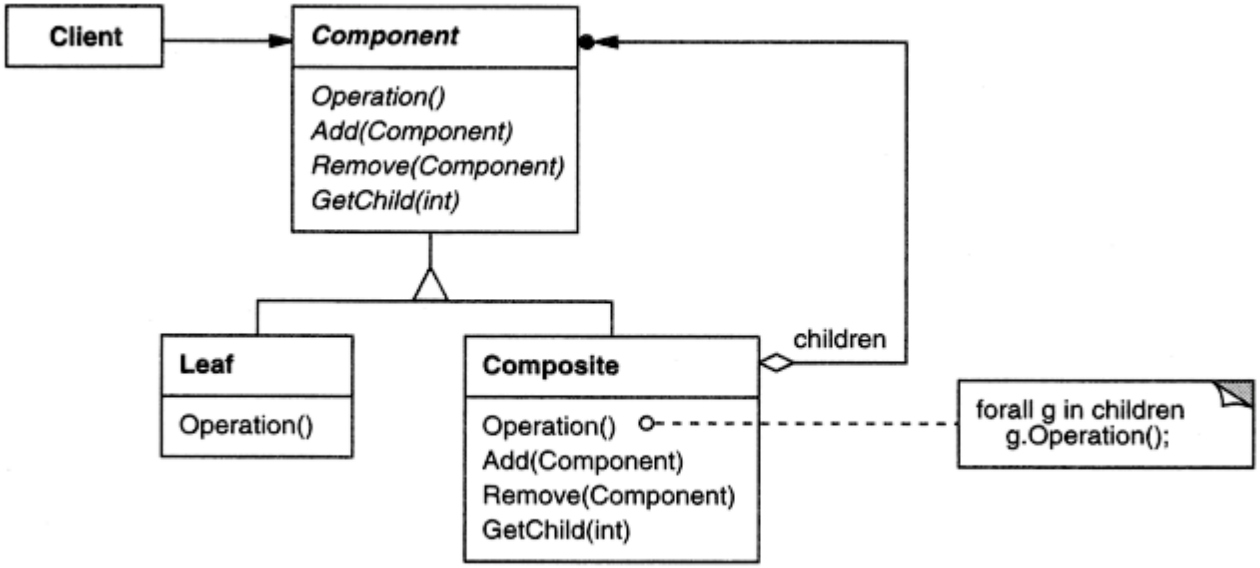

«interface»
ISalePricingStrategy
getTotal(Sale) : Money
PercentDiscount
PricingStrategy
percentage : float
getTotal(s:Sale) : Money
AbsoluteDiscount
OverThreshold
PricingStrategy
discount : Money
threshold : Money
getTotal(s:Sale) : Money
???
PricingStrategy
...
...
{
return s.getPreDiscountTotal() * percentage
}
{
pdt := s.getPreDiscountTotal()
if (pdt < threshold)
return pdt
else
return pdt - discount
}

Composite Design Pattern

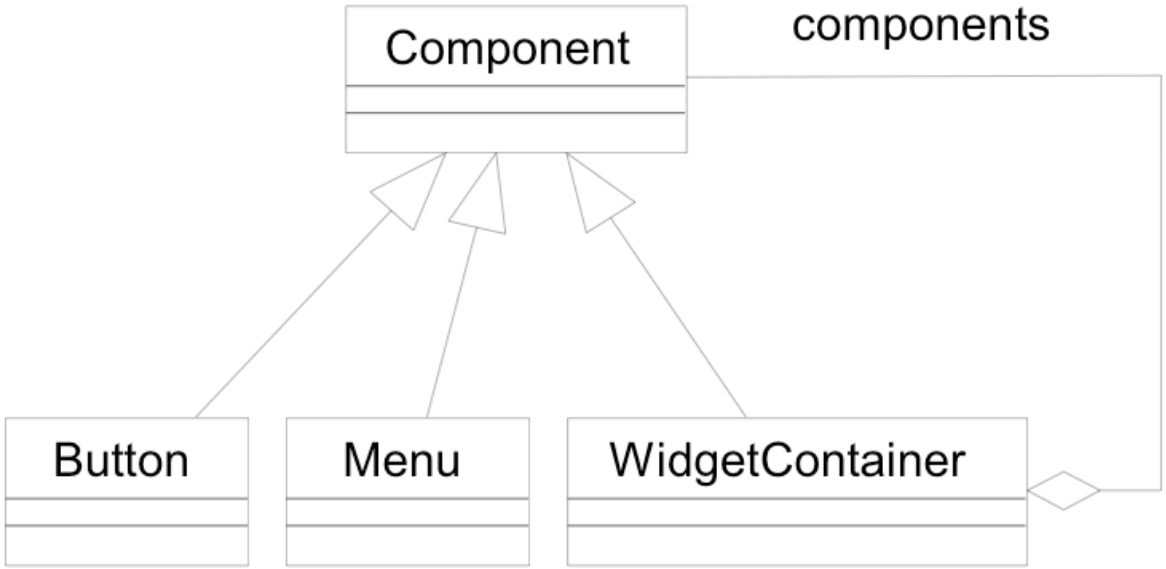
- Context
  - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. This is called *recursive composition*.
- Problem
  - How to treat a group or composition structure of objects the same way (polymorphically) as a non-composite (atomic) object?
- Solution
  - Define classes for composite and atomic objects so that they implement the same interface

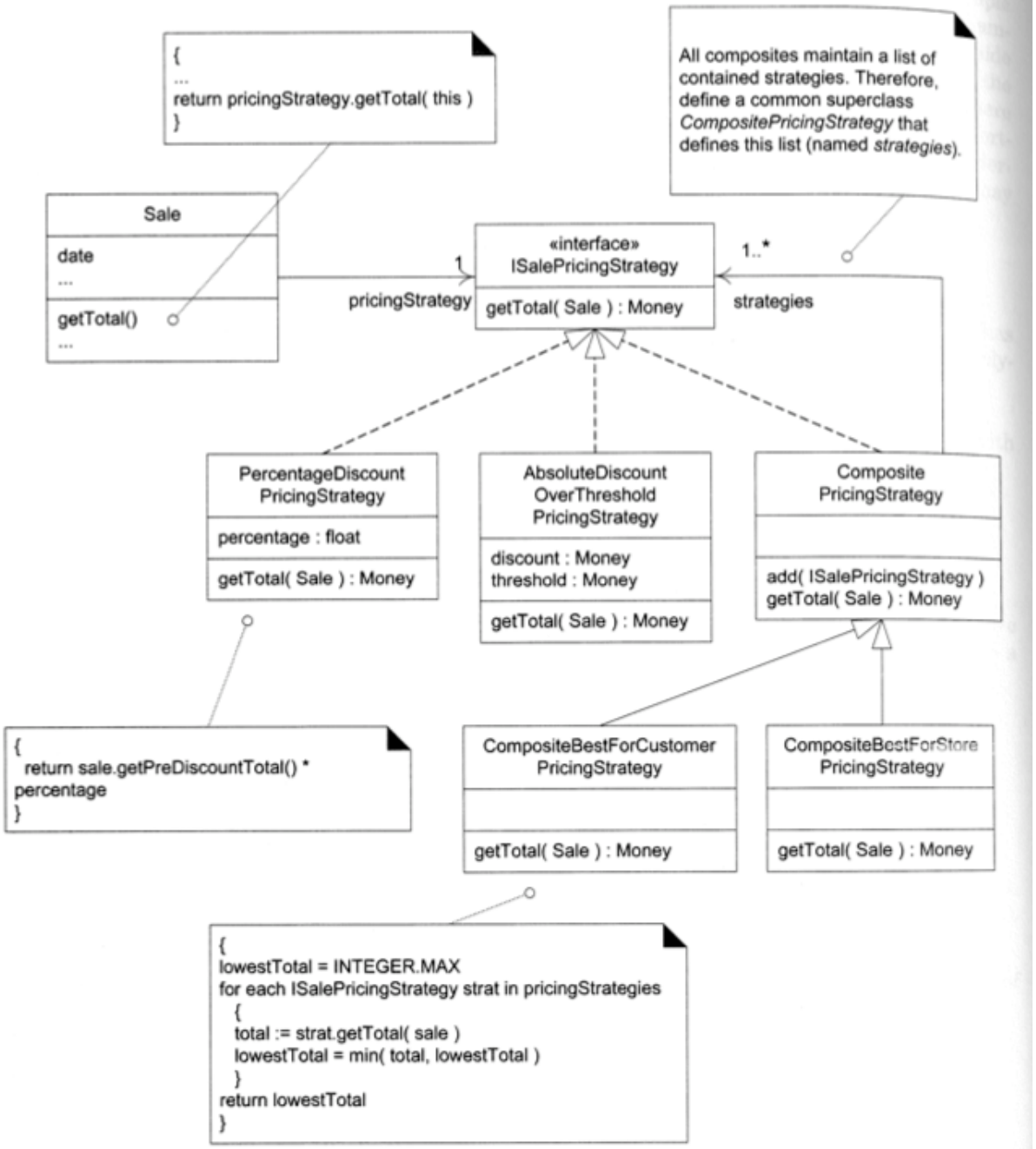
Composite Design Pattern

- Structure



Example




{
...
return pricingStrategy.getTotal(this)
}
All composites maintain a list of contained strategies. Therefore, define a common superclass CompositePricingStrategy that defines this list (named strategies).
Sale
date
...
getTotal()
...
1
pricingStrategy
«interface»
ISalePricingStrategy
getTotal(Sale) : Money
1..*
strategies
PercentageDiscount PricingStrategy
percentage : float
getTotal(Sale) : Money
AbsoluteDiscount OverThreshold PricingStrategy
discount : Money
threshold : Money
getTotal(Sale) : Money
Composite PricingStrategy
add(ISalePricingStrategy)
getTotal(Sale) : Money
{
return sale.getPreDiscountTotal() *
percentage
}
CompositeBestForCustomer PricingStrategy
getTotal(Sale) : Money
CompositeBestForStore PricingStrategy
getTotal(Sale) : Money
{
lowestTotal = INTEGER.MAX
for each ISalePricingStrategy strat in pricingStrategies
{
total := strat.getTotal(sale)
lowestTotal = min(total, lowestTotal)
}
return lowestTotal
}

Composite Design Pattern

- Consequences
  - It makes it easy to add new kinds of components
  - It makes clients simpler, since they do not have to know if they are dealing with a leaf or a composite component
  - It makes it harder to restrict the type of components of a composite

Software Tools & Methods

SOFTWARE PROCESS REVIEW

Software Process

- A Software Process Model is a simplified representation of the software process, presented from a specific perspective.
  - General and abstract
- Software Process is a set of activities whose goal is the development or evolution of software
  - Specific and enacted

- Like the difference between class and object/instance

Dimensions of Variation

- Phased
- Monolithic or Incremental
- Plan-based or adaptive
- One-pass or iterative
- Continuous testing or late testing
- Feedback
- Risk management

Unified Process

- An iterative development process for using object-oriented analysis and design
- Can be applied in a lightweight and agile approach
  - Optionally incorporate XP or Scrum
- Different refinements exist
  - Rational Unified Process (RUP)
  - Agile Unified Process (AUP)

Unified Process

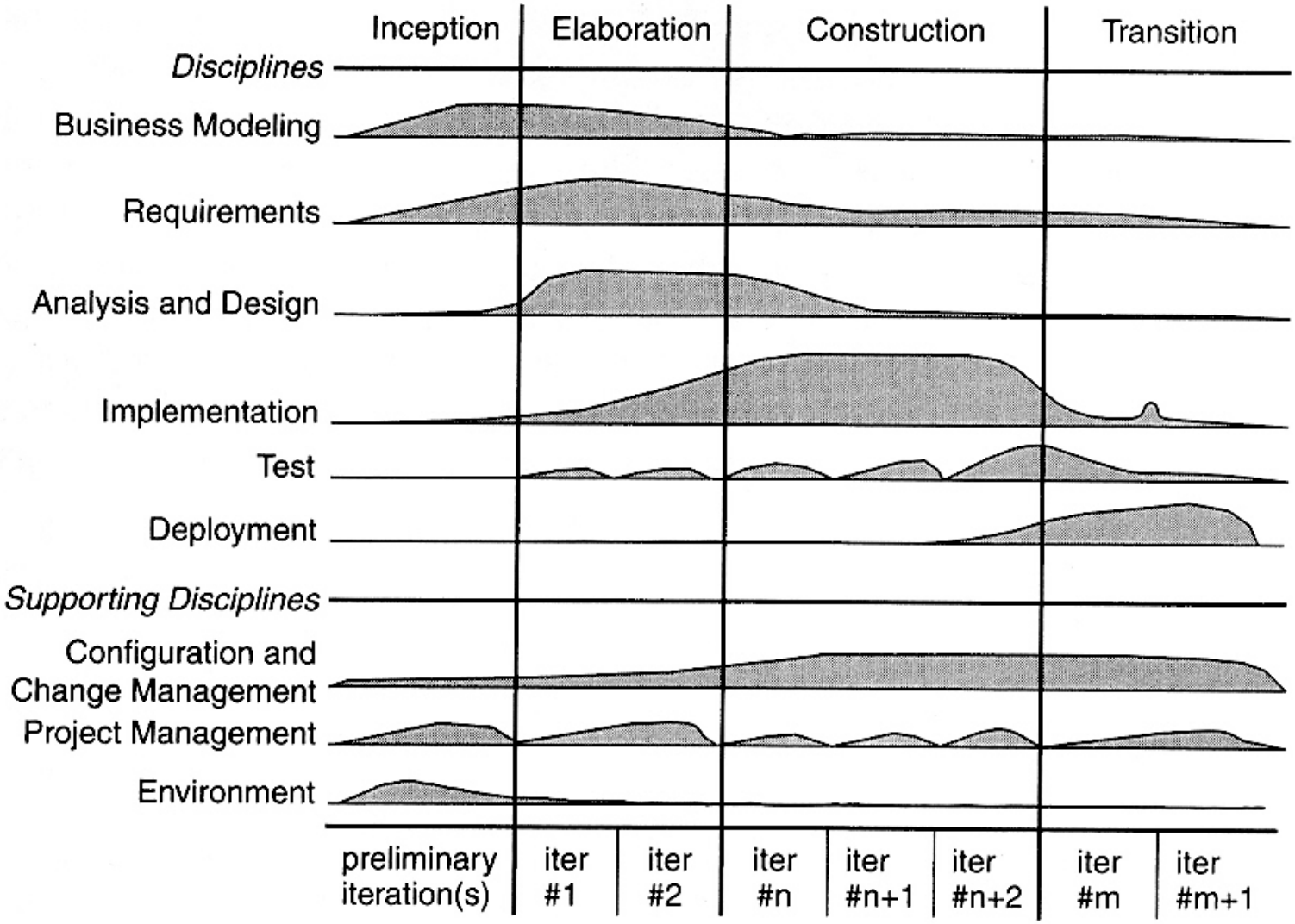
- Iterative
  - Divisions into sub-projects
  - Go through the phases multiple times
- Incremental
  - Builds a sequence of partial systems
- Evolutionary
  - Makes use of feedback from earlier iterations and increments

- Plan to adapt to change, instead of planning to avoid change

Iterative Planning

- Client-driven
  - Build the features that the client cares most about
- Risk-driven
  - Identify and drive down the highest risks
  - Usually means being architecture-centric as well
    - Architecture is the foundation. Having a good one reduces risk.

Rational Unified Process



UP Phases

- Inception
  - Approximate vision, business case, scope, vague estimates
- Elaboration
  - Refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates
- Construction
  - Iterative implementation of lower risk and easier elements, preparation for deployment
- Transition
  - Beta tests, deployment

Agile Methods

- Currently, very popular in industry
- Agile means being able to move quickly
  - Mentally quick and resourceful
- Develop software iteratively and incrementally
- Manage risk by managing scope
  - Strong customer focus
- Continuous feedback
  - Between developers, managers, and customers

Commonly Used Agile Methods

- SCRUM
  - Emphasis is on managing the project
- Extreme Programming (XP)
  - Guides development and management
- Others
  - Lean
  - Crystal

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Agile is not entirely new

- Iterative and incremental process models have existed for a long time
  - Spiral model by Barry Boehm (1985)
- Evolutionary software development
  - Mentioned by Fred Brooks in “No Silver Bullet” (1987)
- Frequent deliveries and feedback
  - EVO by Tom Gilb (1985)

Current Differences

- Popularity
  - Initiated by software developers
  - Now taken up by executives
- New Techniques and Tools
  - Test-driven development
  - Refactoring
  - User stories
- Growing Community with Shared Terminology
- Infrastructure
  - Coaches, training, certification, courses, conferences