

Object Oriented Technology

What, Why, How?

Overview

- Why should I use Object Oriented (OO) Technology?
- What does OO mean?
- How do I get started with OO?
- How does OO affect different areas?
- What standards exist?



Why Object Oriented?

- We need to build systems:
 - That are larger
 - In shorter schedules
 - With fewer defects
 - With fewer people
 - That last for longer
 - That adapt to change well
- Object Oriented technology can *help* us do these things



Characteristics of Bad Designs

- Rigidity
 - Changes are difficult because of dependencies between components
 - Impact of changes propagate throughout system design
- Fragility
 - Changes to a component results in unexpected changes in system behavior
- Immobility
 - A component has dependencies upon a number of unwanted or unnecessary components
 - Cannot extract components for reuse elsewhere without taking many other unwanted components



Object Oriented is the Next Step

- OO concepts are the next stage of evolution of software development
- The principles of OO design can help alleviate the challenges of design
- Historically we see:
 - Increasing levels of abstraction
 - Increasing degrees of modularity
 - Software that is structured and behaves more like the real world
- These are not new ideas, but extensions of existing trends

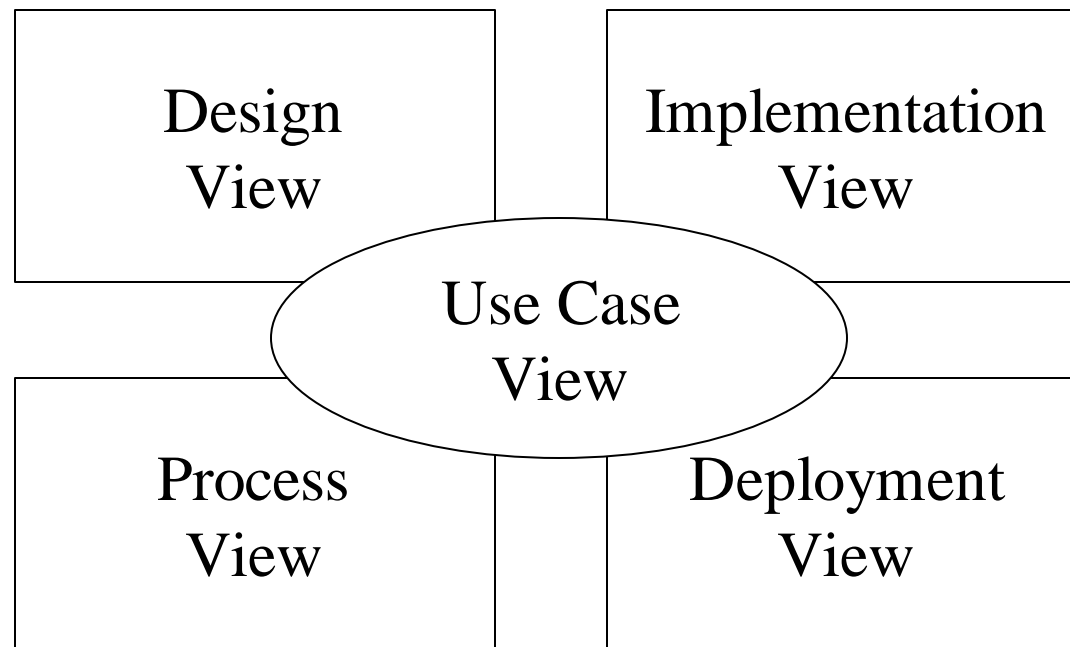


Major Characteristics of OO Systems

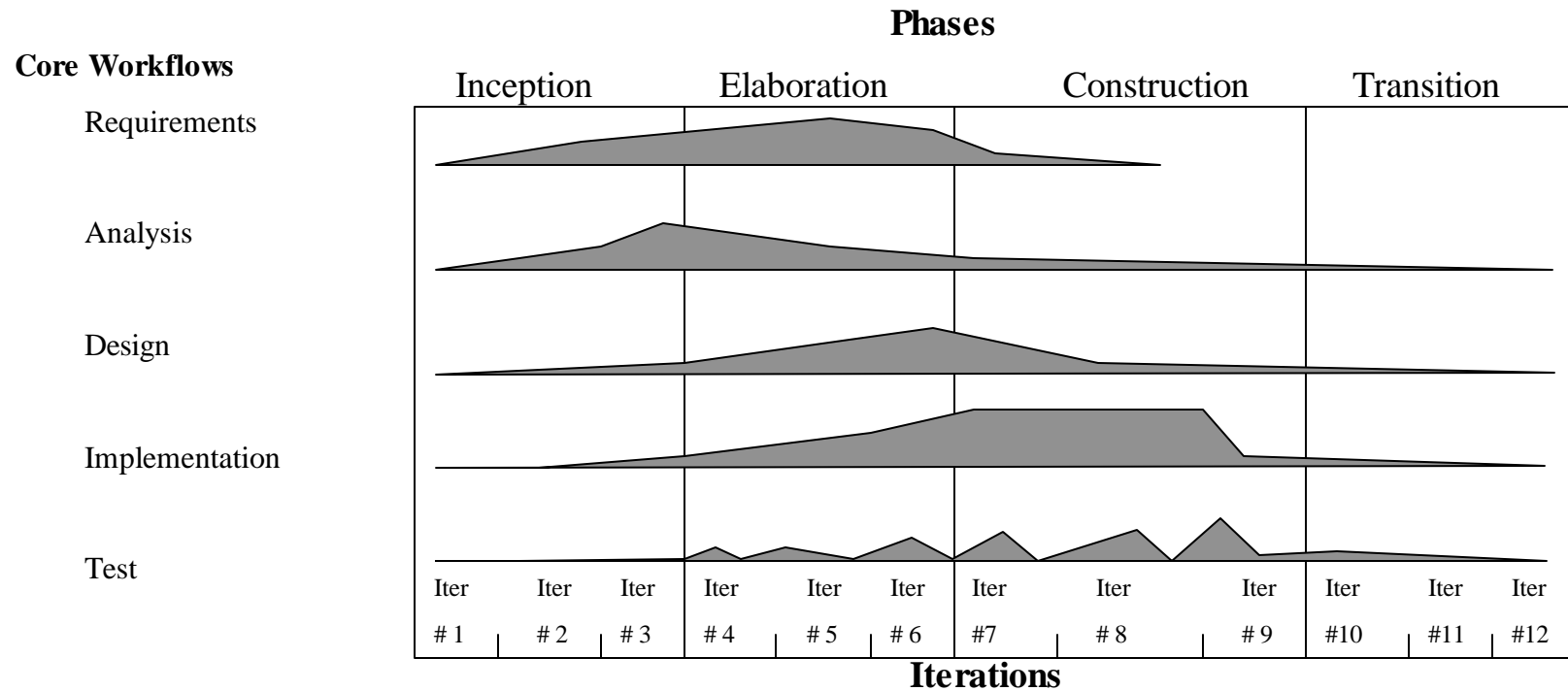
- Objects
- Classes
- Inheritance
- Message Passing
- Polymorphism



4+1 Architectural Views



Unified Process Phases



Objects

- Fundamental characteristic of all OO systems
- Objects are discrete, distinguishable entities
- Often represent physical or conceptual “things”
- Objects replace functions as the fundamental software element in OO systems
- Objects are characterized by
 - Data - Attributes
 - Behavior - Methods (Functions or Operations)
- $\text{Objects} = \text{Data} + \text{Behavior}$



Examples of Objects

- Flight Simulation
 - 737 #314, Lambert Airport
- Payroll Application
 - Joe Smith, Company X, Week 27
- Facility Scheduler
 - The 8:00 AM Friday Status Meeting, Room 222



Object Attributes

- Attributes describe the properties of an object
- Attributes are to objects as fields are to records
- Objects have values for each attribute

Person

name: John
height: 5' 10"
weight: 165 lb..
hair color: brown

Account

owner: Joe Smith
acct #: 123456
balance: 1200.00



Behavior

- The behavior of an object consists of the operations that it can perform
- Behavior Examples:
 - 737 Object: take off, land, autopilot on
 - Employee Object: mail check, file grievance, clock in
 - Vending Machine Object: insert quarter, check coin value, refund
 - Account object: deposit, withdraw, check balance, calculate interest



Classification

- Objects with similar characteristics (attributes and behaviors) are categorized into classes
- The class can be thought of several ways:
 - A set of objects with the same characteristics
 - A template, pattern, or “factory” for creating objects
 - A data type for variables (objects)
- Every object is an instance of some class
- Each object inherently “knows” its class.



Class Examples

- Airplane
- Employee
- Meeting
- F15
- Schedule



Inheritance

- Inheritance allows classes to be defined in terms of other classes
- Objects may share some characteristics but not others
 - Allows objects to reuse existing code (even if the existing class is not exactly what is needed)
- A class which inherits from another class is a subclass or child class
- A subclass inherits attributes and methods from its superclass or parent class
 - The subclass may add its own unique characteristics or customize inherited characteristics.



Inheritance Examples

Parent Class

Employee

Book

Mammal

Airplane

Transaction

Command

Account

Child Class(es)

Hourly Employee, Salaried Employee

Paperback Book

Dog, Cat, Whale

F15, 737, DC-3

Refund Transaction, Sales Transaction

Delete Command, Move Command,
Undo Command, Rename Command

Checking Account, Savings Account

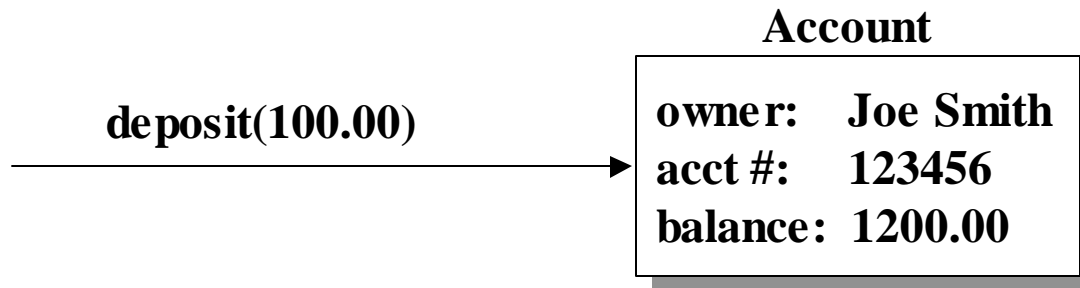
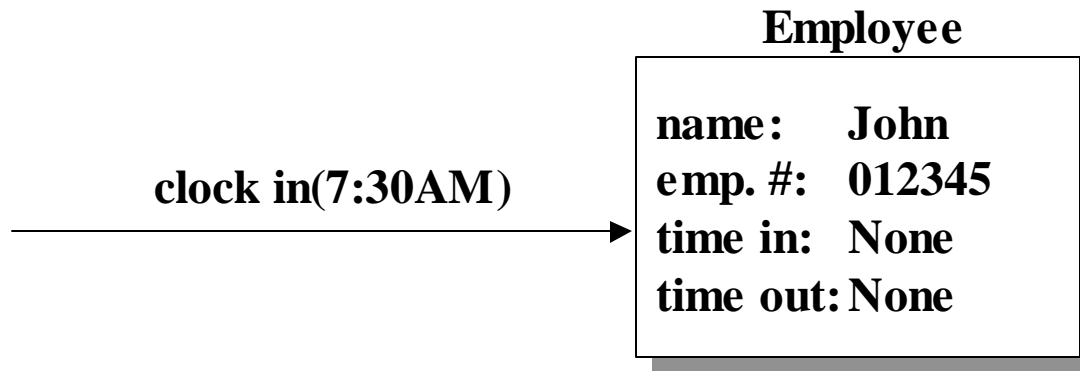


Message Passing

- Objects communicate by passing messages
- A message is identified by a name and a set of parameters
 - *message_name(parameter1, parameter2, ...)*
- When an object receives a message it performs the corresponding method
- Users of the class need not be concerned with the implementation details, only the interface
 - The interface is the set of all messages that a class of objects accepts



Message Passing Examples



Collaboration

- An object oriented system consists of a network of collaborating objects
- Objects do not directly manipulate each other - they send messages to one another
- Each object defines an interface that describes the messages to which it can respond
- Object interfaces are implemented using methods
- Methods can do two things:
 - Change the internal state of the object
 - Coordinate with other objects (send them messages)



Polymorphism

- Function takes on many forms for different objects
- Each object responds to a message in a different way (based on its class)
- Allows different classes of objects to be treated similarly
- Provides abstraction for groups of classes
- Usually enabled by message passing and dynamic binding
 - Dynamic binding is selection of a method at run-time



Polymorphism Example

- There is a family of Employee classes (Employee, Hourly Employee, Sales Employee, Salaried Employee, etc.)
- Each class will have its own calculate pay method
- Users simply send the calculate pay message to employee objects without knowing the kind of Employee it is
- The object “knows” its class and executes the correct method
- New kinds of employees can be added with minimal effect to existing code



How is OO Different?

- Systems are built as models of the world (or problem domain)
 - Existing approaches base the system on the requirements
 - Requirements are more dynamic than our world view
- Encapsulating code and data together into objects gives better modularity
 - Data driven and functional driven approaches tend to result in systems that ignore the opposite viewpoint
 - Object oriented systems attempt to balance these two viewpoints



Advantages of OO

- Systems are easier to maintain and adapt to new requirements
- Provides a consistent paradigm throughout life cycle
- Components are easier to develop separately
- Components are easier to test and integrate
- Reduces downstream errors
- Promotes reuse (analysis, design, and code)
 - Both within and between projects



Potential Problems with OO

- Lack of training - only learn the buzzwords
- More time spent in analysis and design
- Lack of early returns
- Resistance to change (personal and organizational)
- Existing applications are difficult to translate
- Overselling of technology (and resulting backlash)
- Speed and size overhead



Where has OO affected IT?

- Languages
- Analysis/Design
- Databases
- Distributed Systems



Languages

- Existing languages are moving to support OO
 - C (C++, Objective C), Pascal (Object Pascal, Borland Pascal), Ada (Ada 95), Cobol (Object Cobol), Power Builder, etc.
 - These Hybrid approaches support OO and non-OO styles
 - Component based approaches (Visual Basic)
- New OO languages created from scratch
 - Java, Smalltalk, Eiffel, Simula, Trellis, etc.
 - “Force” programmer to use OO concepts



Analysis and Design

- Many new methodologies that support OO
- Same general story as languages (new vs. derived)
- There is a general converging motion in this field
 - Each group is “stealing” the best ideas from others
- The Unified Modeling Language (UML) has emerged as the standard modeling language/notation
- Different methodologies are focused on different types of applications
- Patterns allow for the capture and reuse of successful techniques



Databases

- Relational databases are attempting to provide better support for objects
- Object Oriented Databases (OODB) directly store objects in a database
 - ObjectStore, Objectivity, Versant, Gemstone, Ontos, POET, Jasmine
 - ODMG Standards trying to unify and mature market
- Object-Relational databases try to support both approaches
 - UniSQL, Persistence, others



Distributed Systems

- Goal is to allow objects to interact across language, process, and/or machine boundaries
 - Also allow OO “wrapping” of functional legacy systems
- Traditional approaches are functional (DDE, RPC)
- OO ideas are taking hold in this area
 - OLE/DCOM/ActiveX
 - Microsoft-defined standard (part of Windows)
 - CORBA
 - Defined by the OMG, a consortium of vendors, users, and consultants
 - RMI (Remote Method Invocation)
 - This is a Java-specific technology similar to the above



How do I move towards OO?

- Training
- Pilot Projects
- “Seeding” New projects
- Mentoring/Consulting in targeted areas



Training

- Provide extensive training in OO and any new products being introduced (Languages, Databases, CASE tools)
- OO is evolutionary, but the concepts still require a leap of perspective
- OO is not easy, just better
- Training must be followed by relevant experiences
 - Actual change of perspective usually takes months



Pilot Projects

- Start small and work towards larger projects
- Build an experienced core of individuals
- Provide the necessary hands-on experience
- Reduce the elapsed time until major benefits
- Use the experienced core to seed subsequent projects
- If you can't start with a small project, break a large project into a series of small ones



Mentoring/Consulting

- Growing organic capabilities from scratch is an expensive process
- Use outside expertise to “jump-start” the process
- Target areas where expertise is needed
- Focus experts on growing the necessary skills
- Keep the pilot projects on-track



Standards

- Object Management Group (OMG) Standards
 - Unified Modeling Language (UML) 1.3 - describes a common language/notation for specifying/defining OO systems
 - CORBA 2.3 - series of specifications that allow and enable distributed OO systems
- Languages
 - C++ ISO/ANSI standard released 9/98
 - Java is controlled/licensed/certified by Sun
- Databases
 - Object Database Management Group (ODMG) 3.0 released



Trends

- The term “object oriented” is getting diluted by overuse
- The influence of OO is being felt in all areas of information technology (whether they use the term or not)
- Most markets involve the same battles between OO “upstarts” and established vendors evolving towards OO
- One of the leading areas is in the development of user interfaces

