Software Testing & Quality Assurance

Structured Vs. Object Oriented Approach

Object Oriented Approach

Object-oriented software engineering (OOSE) is an object Modelling language and methodology. Object-Oriented Software Engineering (OOSE) is a software design technique that is used in software design in object-oriented programming.

ABOUT OOSE:

OOSE is developed by **Ivar Jacobson** in 1992. OOSE is the first object-oriented design methodology that employs use cases in software design. OOSE is one of the precursors of the Unified Modeling Language (UML), such as Booch and OMT. **The Booch method** helps to design systems using the object paradigm. It covers the analysis- and design phases of an object-oriented system. **The object-Modelling technique (OMT)** is an object Modelling language for software Modelling and designing.

It includes requirements, an analysis, a design, an implementation and a testing model. Interaction diagrams are similar to UML's sequence diagrams. State transition diagrams are like UML state chart diagrams.

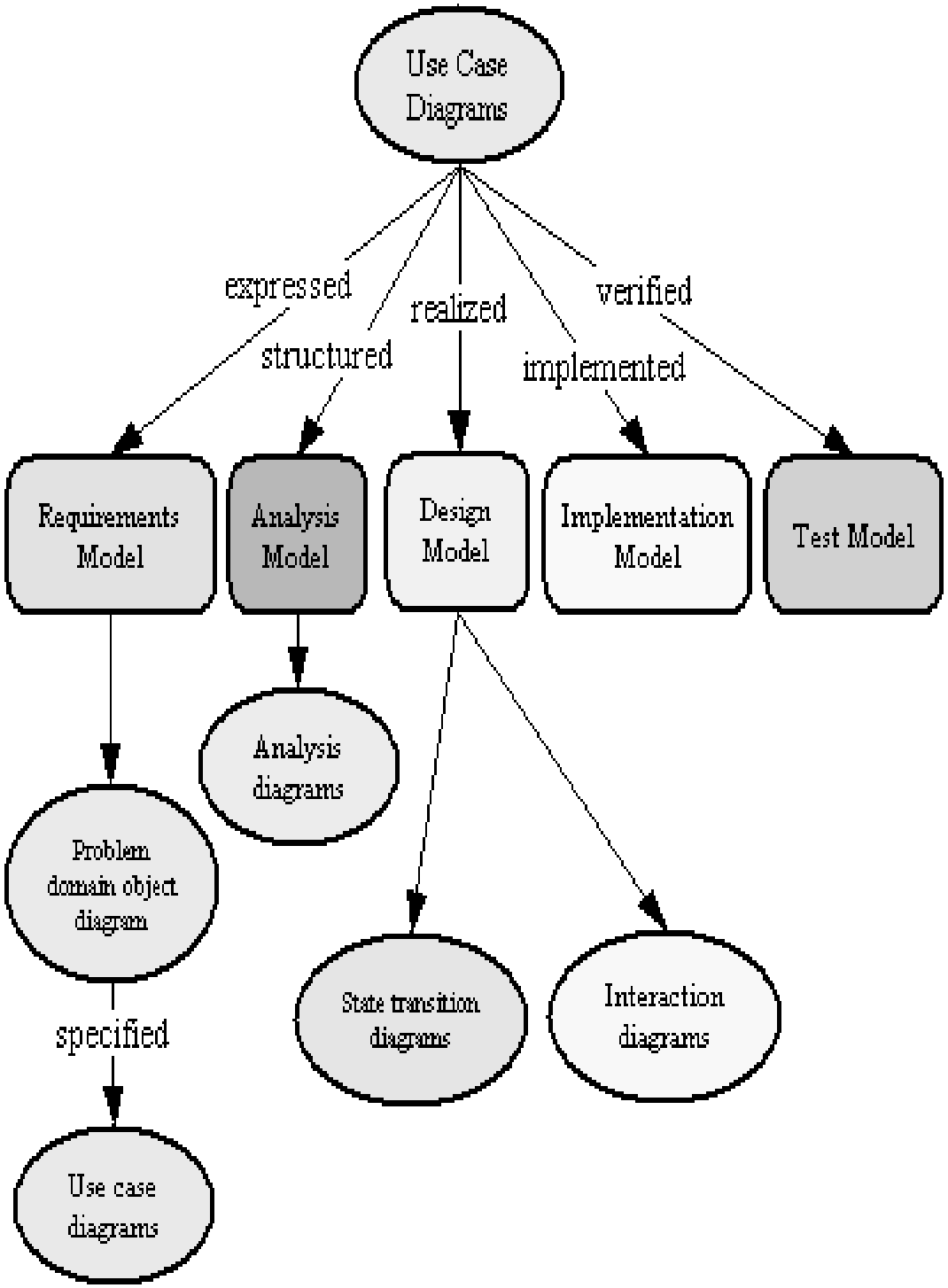


FIGURE 1. Object-Oriented Approach

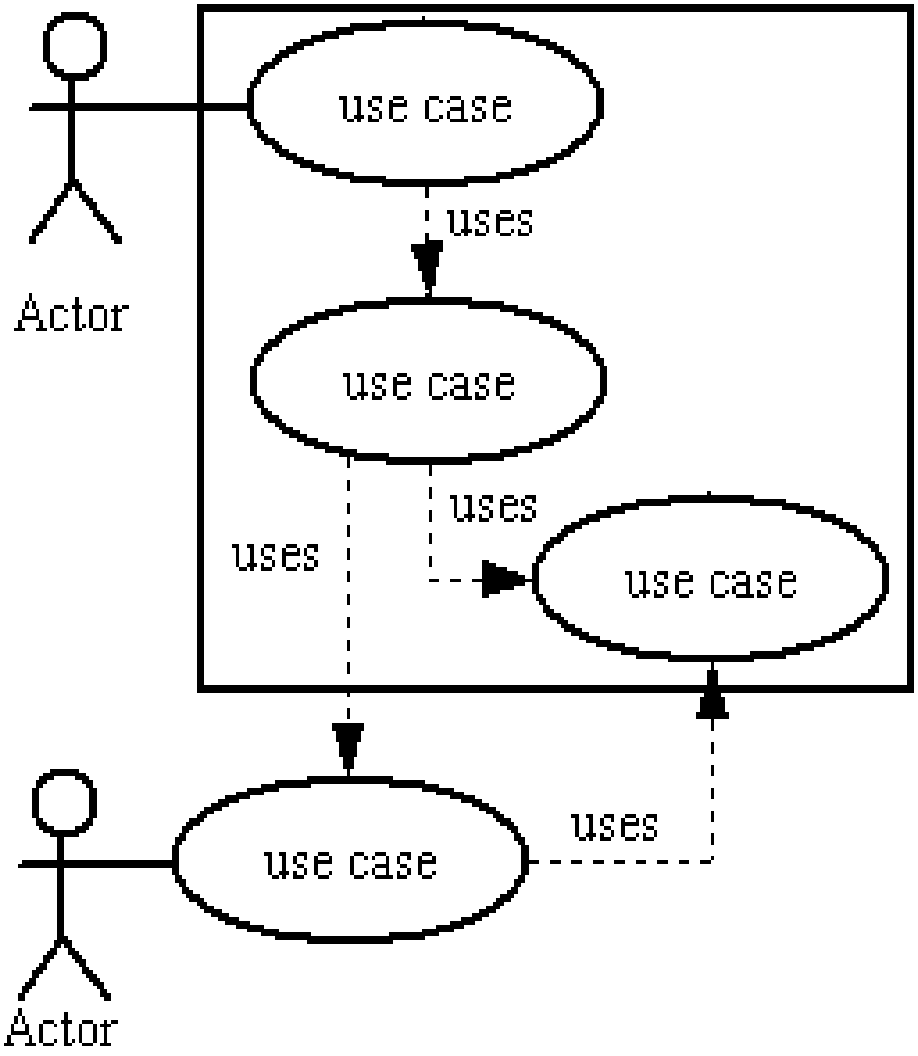


FIGURE 2. JACOBSON'S USE CASE DIAGRAM.

THE STRUCTURED MODEL Vs THE OBJECT ORIENTED MODEL:

STRUCTURED PROGRAMMING:

Structured programming" is programming that links control flow blocks (a function, an if statement's block, etc.) to the scopes of variables. A variable declared inside such a block is invisible outside it.

This leads to programs written with only the following code structures:

1. Conditional execution of statements (i.e., "if" statements).
2. Looping.
3. Structured Subroutine calls (eg. goto statement)
4. Stepwise Refinement

STEPWISE REFINEMENT

Stepwise Refinement is a relatively old technique of Software Design that has been successfully used in a wide range of Structured Programming and Modular Programming environments and languages. It is the procedural (step-by-step) form of SeparationOfConcerns and has what some may call a fractal(curve) nature of task division.

The advantage of Stepwise Refinement is that it allows for Incremental Development but on a much finer level of granularity. Stepwise Refinement is highly scalable, as large systems can be developed in a structured and predictable fashion from it.

Example:

- Brush Teeth
  - find toothbrush
  - find toothpaste tube
  - open toothpaste tube
    - Put thumb and pointer finger on cap
    - turn fingers counter-clockwise
    - repeat prior step until cap falls off
  - clean teeth
    - put brush on teeth
    - move back and fourth vigorously
    - repeat above step 100 times
  - clean up
    - rinse brush
      - turn on water
      - put head of brush under running water for 30 seconds
      - turn off water

- put cap back on toothpaste
- Put all items back in cabinet.

OBJECT-ORIENTED PROGRAMMING:

Object-oriented programming (**OOP**) is a programming paradigm using "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. Many modern programming languages now support OOP.

Many people first learn to program using a language that is not object-oriented. Simple, non-OOP programs may be one long list of commands. More complex programs will group lists of commands into functions or subroutines each of which might perform a particular task.

An object-oriented program will usually contain different types of objects, each type corresponding to a particular kind of complex data to be managed or perhaps to a real-world object or concept such as a bank account, a hockey player. For instance, there could be one bank account object for each real-world account at a particular bank. Each copy of the bank account object would be alike in the methods it offers for manipulating or reading its data, but the data inside each object would differ reflecting the different history of each account.

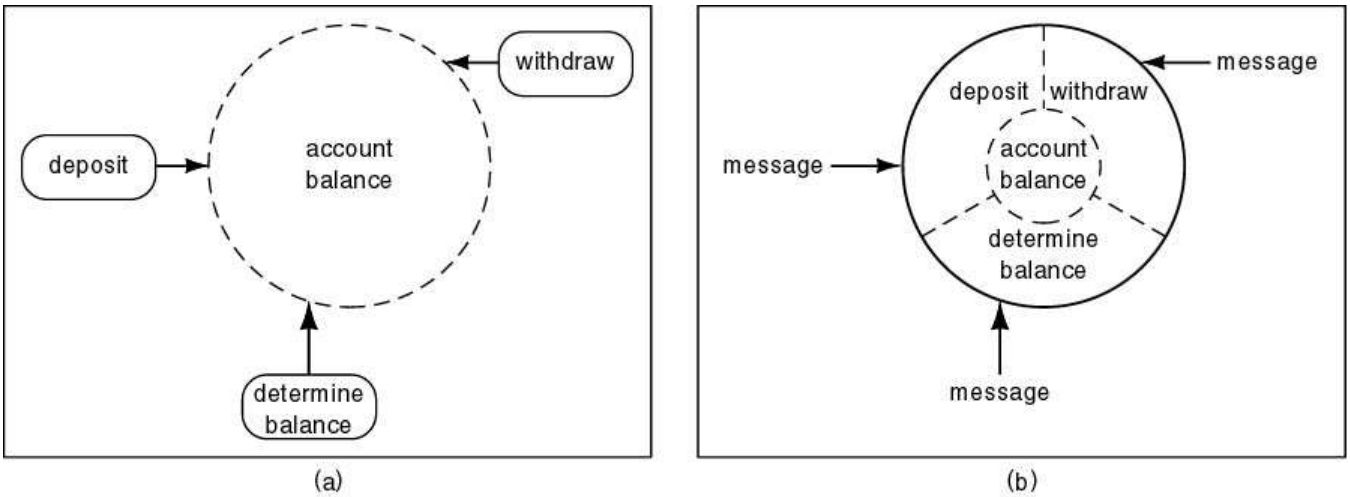
Object-oriented features have been added to many existing languages during that time, including Ada, BASIC, FORTRAN, Pascal, and others. Adding these features to languages that were not initially designed for them often led to problems with compatibility and maintainability of code.

Probably the most commercially important recent object-oriented languages are Visual Basic.NET (VB.NET) and C#, both designed for Microsoft's .NET platform, and Java, developed by Sun Microsystems.

THE OBJECT-ORIENTED PARADIGM:

- The structured paradigm had great successes initially
  - It started to fail with larger products (> 50,000 LOC)
- Maintenance problems (today, up to 80% of effort)
- Reason: structured methods are
  - Action oriented (finite state machines, data flow diagrams); or
  - Data oriented (entity-relationship diagrams, Jackson's method);
  - But not both.
- Both data and actions are of equal importance
- Object:
  - Software component that incorporates both data and the actions that are performed on that data.
  - Example:
    - ✓ Bank account
      - Data: account balance
      - Actions: deposit, withdraw, determine balance

EXAMPLE FOR STRUCTURED VERSUS OBJECT-ORIENTED PARADIGM:



- Information hiding
- Responsibility-driven design
- Impact on maintenance, development

KEY ASPECTS OF OBJECT-ORIENTED SOLUTION:

- Conceptual independence
  - Encapsulation
- Physical independence
  - Information hiding
- Impact on development
  - Physical counterpart
- Impact on maintenance
  - Independence effects.

TRANSITION FROM ANALYSIS TO DESIGN:

Structured Paradigm	Object-OrientedParadigm
1. Requirements phase	1. Requirements phase
2. Specification (analysis) phase	2′. Object-oriented analysis phase
3. Design phase	3′. Object-oriented design phase
4. Implementation phase	4′. Object-oriented programming phase
5. Integration phase	5. Integration phase
6. Maintenance phase	6. Maintenance phase
7. Retirement	7. Retirement

- Structured paradigm:
  - Jolt (change) between analysis (what) and design (how).
- Object-oriented paradigm:
  - Objects enter from very beginning

BENEFITS OF OBJECT-ORIENTATION:

a) Testability/Increased Quality (automated testing can increase speed of testing and increase quality).
b) Code re-use (Polymorphism, Generics, Interfaces)
c) Code extensibility
d) Catch errors at compile time rather than at runtime.
e) Maintainability: If designed correctly, any tier(level) of the application can be replaced by another that implements the correct interface(s), and the application will still work (can use multiple user interfaces).
f) Reduces large problems to smaller, more manageable ones.
g) Fits the way the real world works. It is easy to map a real world problem to a solution in OO code.

Benefits of the object-oriented approach:

Reduced Maintenance: The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated (clearly expressed), the behaviours may be reused and incorporated into new behaviours.

Real-World Modelling: Object-oriented system tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and objects are associated with behaviours. The model is based on objects, rather than on data and processing.

Improved Reliability and Flexibility: Object-oriented system promise to be far more reliable than traditional systems, primarily because new behaviours can be "built" from existing objects. Because objects can be dynamically called and accessed, new objects may be created at any time. The new objects may inherit data attributes from one, or many other objects. Behaviours may be inherited from super-classes, and novel behaviours may be added without effecting existing systems functions.

High Code Reusability: When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also

inherit the data and behaviours from all superclasses in which it participates. When a user creates a new type of a widget, the new object behaves "wigitty", while having new behaviors which are defined to the system.

DRAWBACKS OF OBJECT-ORIENTATION:

- **Object-oriented Development is not a panacea** - Object-oriented Development is best suited for dynamic, interactive environments, as evidenced by its widespread acceptance in CAD/CAM and engineering design systems. Wide-scale object-oriented corporate systems are still unproved, and many bread-and-butter information systems applications (i.e. payroll, accounting), may not benefit from the object-oriented approach.
- **Object-oriented Development is not yet completely accepted by major vendors -** Object-oriented Development has gained some market respectability, and vendors have gone from catering to a "lunatic fringe" to a respected market. Still, there are major reservations as to whether Object-oriented development will become a major force, or fade(gradually grow and disappear) into history, as in the 1980's when Decision Support Systems made great promises, only to fade into obscurity(full details).
- **Cannot find qualified programmers and DBA's-** When one investigates the general acceptance of object-oriented systems in the commercial marketplace, you generally find that most managers would like to see an object technology approach, but they do not have the time to train their staffs in object-oriented methods. Other will say that the object-oriented method is only for graphical workstation systems, and that there is no pressing need for object-oriented system within mainstream business systems.

****