Object Oriented Software Development Life Cycle

The Software Development Process

- Why a Process?
  - Software projects are large, complex, sophisticated
  - Time to market is key
  - Many facets involved in getting to the end
- Common process should
  - Integrate the many facets
  - Provide guidance to the order of activities
  - Specify what artifacts need to be developed
  - Offer criteria for monitoring and measuring a project

Software Development Process

- Steps correspond to one or more tasks related to software development.
- Tasks:
  - Requirements gathering
  - Requirements analysis
  - Design
  - Coding
  - Integration
  - Test
  - Delivery
  - Maintenance
  - Training

Software Life Cycle: Software Life Cycle consists of all phases from its inception until its retirement. These are (for Unified Process): Inception, elaboration, construction, transition.

Why a Process?

- Software projects are large, complex, sophisticated
- Time to market is key
- Many facets involved in getting to the end

Common process should

- Integrate the many facets
- Provide guidance to the order of activities
- Specify what artifacts need to be developed
- Offer criteria for monitoring and measuring a project

A good software process is repeatable, predictable, learnable, measurable and improvable.

The Software Development Process

- ’It is better not to proceed at all, than to proceed without method.' — Descartes
- The Software Development Process:
  - The framework for the set of tasks that are required to develop a software system.
  - Process defines how a software product is developed and maintained.
  - A well-defined and rigorously enforced process forms the basis for high-quality software development.

Having a defined process is essential

- Life cycle is the series of steps that software undergoes from concept exploration through retirement

Maturity of the process is some gauge of success of organization

Software Life Cycle Model

- Definition
  - Describes an abstract collection of software processes that share common characteristics such as timing between phases, entry and exit criteria for phases.
- The models specifies
  - The various phases of the process
    - E.g., requirements, specification, design...
    - The order in which they are carried out
Importance of Lifecycle Models

- Provide guidance for project management
  - what major tasks should be tackled next? milestones!
  - what kind of progress has been made?
- The necessity of lifecycle models
  - character of software development has changed
    - early days: programmers were the primary users
    - modest designs; potential of software unknown
    - more complex systems attempted
      - more features, more sophistication → greater complexity,
        more chances for error
      - heterogeneous users

Life Cycle Models

- **Build and fix**: Acceptable for short programs that do not require maintenance.
- **Waterfall**: Disciplined approach, document driven; delivered product may not meet client needs.
- **Rapid prototyping**: Ensures that delivered product meets client needs; might become a build-and-fix model.
- **Incremental**: Maximizes early return on investment; requires open architecture; may degenerate into build-and-fix.

Object-Oriented Life-Cycle Models

- Need for iteration within and between phases
  - Fountain model
  - Recursive/parallel life cycle
  - Unified software development process
- All incorporate some form of
  - Iteration
  - Parallelism
  - Incremental development
- Danger
  - CBTAB

Unified Development Process [1]

- Key features: Iterative development; OO analysis and design.
- Development organized as a series of short iterations
- Each iteration produces a working, executable product that *might not be a deliverable*.
- No rush to code. Also, not a long drawn design process.
- Lots of visual modeling aids. Unified Modeling Language (UML) used.

Unified Development Process [2]

- Early iterations seek feedback from the customer.
  - Risk and value to customer is managed through early feedback.
- Customer is engaged continuously in evaluation and requirements gathering.
  - Architecture is built during early iterations.
Unified Development Process [3]

The Unified Process

- Component based – meaning the software system is built as a set of software components interconnected via interfaces
- Uses the Unified Modeling Language (UML)
- Use case driven
- Architecture-centric
- Iterative and Incremental

Component: A physical and replaceable part of a system that conforms to and provides realization of a set of interfaces.
Interface: A collection of operations that are used to specify a service of a class or a component

User's requirements
Software Development Process
Software System

Based around the 4Ps - People, Project, Product, Process

The Unified Process

- Use Case driven
  - A use case is a piece of functionality in the system that gives a user a result of value.
  - Use cases capture functional requirements
  - Use case answers the question: *What is the system supposed to do for the user?*

The Unified Process

- Architecture centric
  - similar to architecture for building a house
  - Embodies the most significant static and dynamic aspects of the system
  - Influenced by platform, OS, DBMS etc.
  - Related as *function* (use case) and *form* (architecture)
  - Primarily serves the realization of use cases
  - The form must allow the system to evolve from initial development through future requirements (i.e. the design needs to be flexible)

The Unified Process

- Iterative and Incremental
  - commercial projects continue many months and years
  - to be most effective – break the project into *iterations*
  - Every iteration – identify use cases, create a design, implement the design
  - Every iteration is a complete development process

This is what makes the Unified process Unique

Component: A physical and replaceable part of a system that conforms to and provides realization of a set of interfaces.
Interface: A collection of operations that are used to specify a service of a class or a component
An iterative and incremental process

![Diagram](image)

**Iterations**

- Iterations must be selected & developed in a planned way i.e. in a logical order – early iterations must offer utility to the users
  - iteration based on a group of use cases extending the usability of the system developed so far
  - iterations deal with the most important risks first
  - not all iterations are additive – some replace earlier “superficial” developments with a more sophisticated and detailed one.

**Goals**

- Object-Oriented Methodologies
  - The Rumbaugh et al. OMT
  - The Booch methodology
  - Jacobson's methodologies

**Basic Definitions**

- A methodology is explained as the science of methods.
- A method is a set of procedures in which a specific goal is approached step by step.

**Too Many Methodologies**

- 1986: Booch came up with the object-oriented design concept, the Booch method.
- 1987: Sally Shlaer and Steve Mellor came up with the concept of the recursive design approach.
Too Many Methodologies (Con’t)

- 1989: Beck and Cunningham came up with class-responsibility-collaboration (CRC) cards.
- 1990: Wirfs-Brock, Wilkerson, and Wiener came up with responsibility-driven design.
- 1991: Peter Coad and Ed Yourdon developed the Coad lightweight and prototype-oriented approach.

Implication

- Split, Competetive & Confusing
- Same basic concepts in different notations
- Second-generation oo methodologies – combining best aspects

Survey of Some of the Object-Oriented Methodologies

- Many methodologies are available to choose from for system development.
- Here, we look at the methodologies developed by Rumbaugh et al., Booch, and Jacobson which are the origins of the Unified Modeling Language (UML) and the bases of the UA.

Rumbaugh et. al.’s Object Modeling Technique (OMT)

- OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique.

OMT (Con’t)

- OMT consists of four phases, which can be performed iteratively:
  - Analysis. The results are objects and dynamic and functional models.
  - System design. The result is a structure of the basic architecture of the system.
  - Object design. This phase produces a design document, consisting of detailed objects and dynamic and functional models.
  - Implementation. This activity produces reusable, extendible, and robust code.
OMT Modeling

- OMT separates modeling into three different parts:
  - 1. An **object model**, presented by the object model and the data dictionary.
  - 2. A **dynamic model**, presented by the state diagrams and event flow diagrams.

Object Model

- Describes structure of objects in system
  - Identity
  - Relationship to other objects
  - Attributes
  - Operations

- Represented graphically with object diagram
  - Contains classes interconnected by association lines
    - Each class represents a set of individual objects
    - Association line represents set of links from objects of one class to objects of another class

OMT Object Model of a Bank System

**FIGURE 4.1**
The OMT object model of a bank system. The boxes represent classes and the filled triangle represents a specialization relationship between classes and classes of the same type. The solid lines represent association lines between objects and classes. An association line represents a set of links from objects of one class to objects of another class.

OMT Dynamic Model

- OMT state transition model is a network of states & events
  - Depict states, transitions, events, actions
  - Each state receives one or more events at which time it makes the transition to the next state
  - Next state depends on events + current state

**FIGURE 4.2**
State transition diagram for the bank application user interface. The round boxes represent states and the arrows represent transitions.

OMT Functional Model

- Describes business process without focusing on computer systems details
- OMT data flow diagram shows flow of data between different processes in business
  - Processes – function performed
  - Data flow – direction of data element movement
  - Data store – location where data are stored
  - External entity – source/destination of data element

**FIGURE 4.3**
Data flow diagram for the bank application user interface. The round boxes represent nodes and the arrows represent data flow.
Analysis & design method using object oriented paradigm
- Start with class & object diagrams in analysis phase
- Diagrams refined in various steps
- Add design symbols when ready to generate code
- Object-oriented code documented

Diagrams in Booch Method
- Class diagrams
- Object diagrams
- State transition diagrams
- Module diagrams
- Process diagrams
- Interaction diagrams

Eg. of Booch class diagram

Booch’s Macro Development Process

Conceptualization
- Establish core requirements
- Establish goal
- Develop prototype to prove concepts
- Analysis & development of model
- Use class diagram to describe roles & responsibilities of objects in system
- Use object diagram or interaction diagram to describe desired behavior of system in terms of scenarios

Design & create system architecture
- Use class diagram to decide what classes exist, how they relate to one another
- Use object diagram to decide what mechanisms are used to regulate how objects collaborate
- Use module diagram to map where each class & object should be declared
- Use process diagram to determine process allocation to processors
- Determine schedule for multiple processes on processors

Evolution or Implementation
- Refine system through iterations
- Produce stream of software implementations/executable releases

Maintenance
- Make localized changes to add new requirements/eliminate bugs
Booch’s Micro Development Process

- Day to day activities for each macro development process
- Can consists of following steps
  - Identify classes & objects
  - Identify Class & object semantics
  - Identify class & object relationships
  - Identify class & object interfaces/implementation

Jacobson et al. Methodologies

- Object-oriented business engineering (OOBE)
- Object-oriented software engineering (OOSE)
- Object Factory for Software Development (Objectory)
- Cover entire life cycle
- Stress traceability between phases – backward & forward
- Enables reuse of analysis & design work
- Use-case concepts used

Use cases

- Scenarios for understanding system requirements
- Use case is interaction between users & system
- Captures goal of users & responsibility of system to users

Eg of Library use case

- Use case driven development
  - Use case used in analysis, design, validation, testing
  - Scenarios begins with a user initiating sequences of interrelated events
- Built around several models
  - Use case model – define actors & inside system behavior
  - Domain object model – map objects in real world
  - Analysis object model – presents how source code should be carried out & written
  - Implementation model – represent system implementation
  - Test model – constitute test plan, specification & reports

In requirements analysis, describe use case as

- Nonformal text
- Text with clear flow of events
- Formal style using pseudo code
- Description must contain
  - How & when use case begin
  - Interaction between use case & its actors
  - When interaction occurs, what is exchanged
  - How use case will need data stored in system, will store data in system
  - Exceptions to flow of events
  - How & when concepts of problem domain are handled

Object-Oriented Software Engineering
The Unified Approach

- The idea behind the UA is not to introduce yet another methodology.
- The main motivation here is to combine the best practices, processes, methodologies, and guidelines along with UML notations and diagrams.

The Unified Approach (UA)

- The unified approach to software development revolves around (but is not limited to) the following processes and components.
- The processes are:
  - Use-case driven development.
  - Object-oriented analysis.
  - Object-oriented design.
  - Incremental development and prototyping.
  - Continuous testing.

UA Methods and Technology

- The methods and technology employed includes:
  - Unified modeling language (UML) used for modeling.
  - Layered approach.
  - Repository for object-oriented system development patterns and frameworks.
  - Promoting Component-based development.
UA Object-Oriented Analysis: Use-Case Driven

- The use-case model captures the user requirements.
- The objects found during analysis lead us to model the classes.
- The interaction between objects provide a map for the design phase to model the relationships and designing classes.

UA Object-Oriented Design

- Booch provides the most comprehensive object-oriented design method.
- However, Booch methods can be somewhat imposing to learn and especially tricky to figure out where to start.
- UA realizes this by combining Jacobson et al.’s analysis with Booch’s design concept to create a comprehensive design process.

Iterative Development and Continuous Testing

- The UA encourages the integration of testing plans from day 1 of the project.
- Usage scenarios or Use Cases can become test scenarios; therefore, use cases will drive the usability testing.

Modeling Based on the Unified Modeling Language

- The UA uses the unified modeling language (UML) to describe and model the analysis and design phases of system development.

The UA Proposed Repository

- The requirement, analysis, design, and implementation documents should be stored in the repository, so reports can be run on them for traceability.
- This allows us to produce designs that are traceable across requirements, analysis, design, implementation, and testing.

The Layered Approach to Software Development

- Most systems developed with today’s CASE tools or client–server application development environments tend to lean toward what is known as two-layered architecture: interface and data.
Two-Layer Architecture

- In a two-layer system, user interface screens are tied directly to the data through routines that sit directly behind the screens.

Problem With the Two-Layer Architecture

- This approach results in objects that are very specialized and cannot be reused easily in other projects.

Three-Layer Architecture

- Your objects are completely independent of how:
  - they are represented to the user (through an interface) or
  - how they are physically stored.

User Interface layer

This layer is typically responsible for two major aspects of the applications:

- Responding to user interaction
- Displaying business objects

Business Layer

- The responsibilities of the business layer are very straightforward:
  - model the objects of the business and how they interact to accomplish the business processes.

Business Layer: Real Objects (Con’t)

These objects should not be responsible for:

- Displaying details
- Data access details
Access Layer

- The access layer contains objects that know how to communicate with the place where the data actually resides.
- Whether it be a relational database, mainframe, Internet, or file.

Three-Layered Architecture

Summary

- We looked at current trends in object-oriented methodologies, which have been toward combining the best aspects of today's most popular methods.

Summary (Con't)

- Each method has its strengths. Rumbaugh et al. have a strong method for producing object models.
- Jacobson et al. have a strong method for producing user-driven requirement and object-oriented analysis models.
- Booch has a strong method for producing detailed object-oriented design models.

Summary (Con't)

- Each method has weakness, too. While OMT has strong methods for modeling the problem domain, OMT models cannot fully express the requirements.
- Jacobson, although covering a fairly wide range of the life cycle, does not treat object-oriented design to the same level as Booch, who focuses almost entirely on design, not analysis.
Summary (Con’t)

› The UA is an attempt to combine the best practices, processes, and guidelines along with UML notations and diagrams for better understanding of object-oriented concepts and object-oriented system development.

Unified Software Development Process (UP)

Selects from best practices to

- Provide a generic process framework
  - instantiate/specialize for specific application areas, organizations, project sizes, etc.

- Define a set of activities (workflows)
  - transforms users’ requirements into a software system

- Define a set of models
  - from abstract (user-level) to concrete (code)

- Allow component-based development
  - software components interconnected via well-defined interfaces
  - use-case (UML) and risk driven
  - architecture-centric
  - iterative and incremental

Unified Process – Milestones

Milestone: a management decision point in a project that determines whether to authorize movement to the next iteration/phase

Inception phase – agreement among customers/developers on the system’s life cycle objectives

Elaboration phase – agreement on the viability of the life cycle architecture, business case and project plan

Construction phase – agreement on the acceptability of the software product both operationally and in terms of cost

Transition phase – final agreement on the acceptability of the software product

Timeboxing

Management of a UP project.

Iterations are “timeboxed” or fixed in length.

Iteration lengths of between two to six weeks are recommended.

Each iteration period has its own development plan.

If all the planned activities cannot be completed during an iteration cycle, the completion date should not be extended, but rather tasks or requirements from the iteration should be removed and added to the next iteration cycle.

Benefits of an iterative approach

› Risks are mitigated earlier
› Change is more manageable
› Higher level of reuse
› Project team can learn along the way
› Better overall quality
The Unified Software Development Process is a definition of a complete set of activities to transform users' requirements through a consistent set of artifacts into a software product.

Look at the whole process:
- Life cycle
- Artifacts
- Workflows
- Phases
- Iterations

A process is described in terms of workflows where a workflow is a set of activities with identified artifacts that will be created by those activities.

The Unified process repeats over a series of cycles.
Each cycle concludes with a product release.
Each cycle consists of four phases:
- Inception - Define the scope of project
- Elaboration - Plan project, specify features, baseline architecture
- Construction - Build the product
- Transition - Transition the product into end user community

The Life of the Unified Process

Inception Phase
- Purpose:
  - To establish the business case for a new system or for a major update of an existing system
  - To specify the project scope
- Outcome:
  - A general vision of the project’s requirements, i.e., the core requirements
  - Initial use-case model and domain model (10-20% complete)
  - An initial business case, including:
    - Success criteria (e.g., revenue projection)
    - An initial risk assessment
    - An estimate of resources required
- Milestone: Lifecycle Objectives

Elaboration Phase
- Purpose:
  - To analyze the problem domain
  - To establish a sound architectural foundation
  - To address the highest risk elements of the project
  - To develop a comprehensive plan showing how the project will be completed
- Outcome:
  - Use-case and domain model 80% complete
  - An executable architecture and accompanying documentation
  - A revised business case, incl. revised risk assessment
  - A development plan for the overall project
- Milestone: Lifecycle Architecture

Construction Phase
- Purpose:
  - To incrementally develop a complete software product which is ready to transition into the user community
- Products:
  - A complete use-case and design model
  - Executable releases of increasing functionality
  - User documentation
  - Deployment documentation
  - Evaluation criteria for each iteration
  - Release descriptions, including quality assurance results
  - Updated development plan
- Milestone: Initial Operational Capability
Transition Phase

- **Purpose**
  - To transition the software product into the user community
- **Products**
  - Executable releases
  - Updated system models
  - Evaluation criteria for each iteration
  - Release descriptions, including quality assurance results
  - Updated user manuals
  - Updated deployment documentation
  - "Post-mortem" analysis of project performance
- **Milestone: Product Release**

---

**Agile Principles**

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently.
4. Customers and developers work together daily.
5. Give individuals the environment and support they need to get job done, and trust them to do it. Build projects around motivated individuals.
6. The most effective way of conveying information to and within a group is face-to-face.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
10. Continuous attention is paid to technical excellence and good design.
11. Simplicity - the art of maximizing the amount of work NOT done - is essential.
12. The best architectures, requirements, and designs emerge from self-organizing teams.
13. At regular intervals, the team reflects on how to become more effective and adapts accordingly.

---

**Agile Processes**

<table>
<thead>
<tr>
<th>The Agile Manifesto</th>
<th>over processes and tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals and interactions</td>
<td></td>
</tr>
<tr>
<td>Working software</td>
<td>over comprehensive documentation</td>
</tr>
<tr>
<td>Customer collaboration</td>
<td>over contract negotiation</td>
</tr>
<tr>
<td>Responding to change</td>
<td>over following a plan</td>
</tr>
</tbody>
</table>

---

**The Rules and Practices of Extreme Programming**

**Planning**
- User stories are written.
- Release plan creates the schedule.
- A project velocity is measured.
- The project is divided into iterations.
- A prioritization meets each iteration.
- A stand-up meeting starts each day.
- Fix it when it breaks.

**Coding**
- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair writes code at a time.
- Integrate before.
- Use collective code ownership.
- Leave the codebase in a better state.
- No estimate.

**Designing**
- Complexity.
- Choose a system metaphor.
- The CRC cards for design reviews.
- Create agile refactoring to reduce risk.
- No functionality is added later.
- Reflect, whatever and whenever possible.

**Testing**
- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found, tests are created.
- Accurate tests are not obvious, and the tests are published.

---

http://www.extremeprogramming.org/rules.html