Software Development Life Cycle

Paul Fodor

CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

Topics

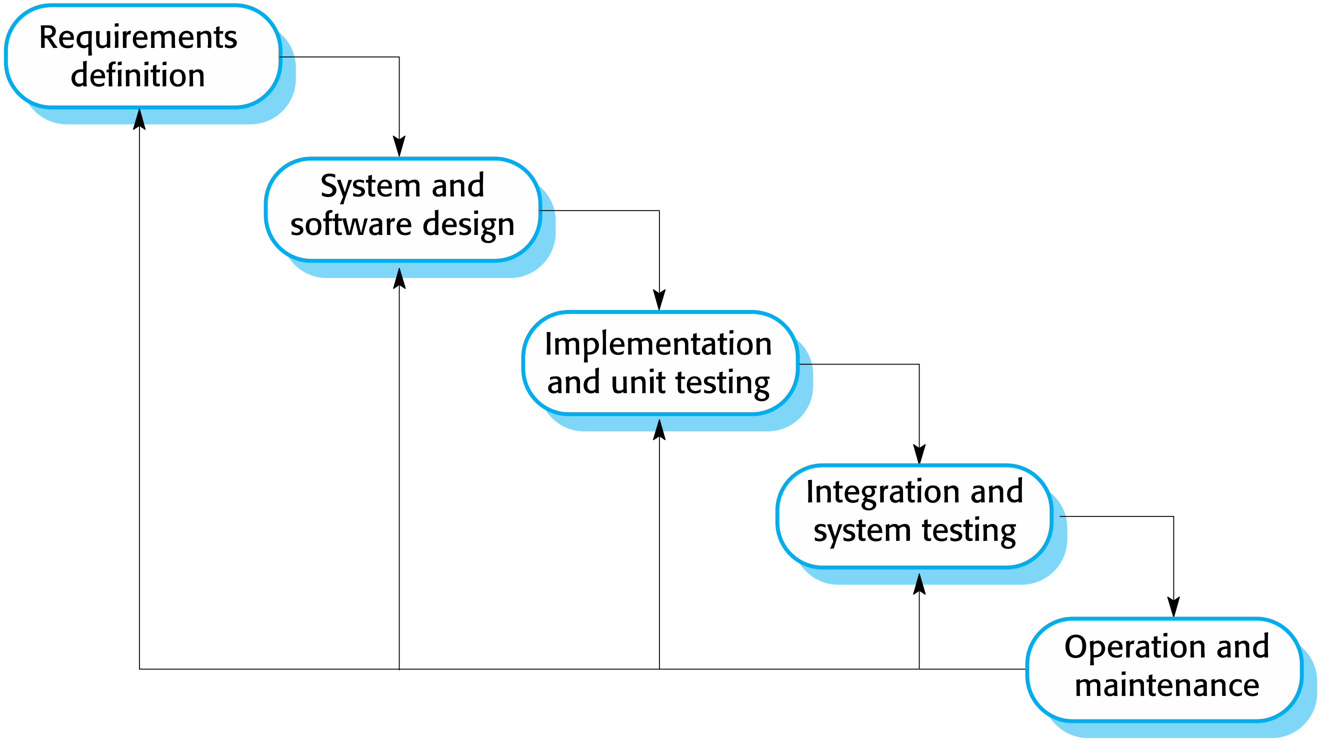
- Overview of the Software Development Life Cycle
- Process Models
- Standard stages:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance

The Software Development Life Cycle

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do (requirements)
  - Design – Architecture of the system (high level design)
  - Detailed Design – Design of component modules, data structures, algorithms, etc.
  - Implementation –Implementing (Coding and Testing) the system
  - Validation (Testing) – Checking that code works and it does what the customer wants
  - Deployment – Putting the system in production
  - Evolution (Optional) – Changing the system in response to changing customer needs.

The Waterfall Model

- Plan-driven model.
  - Specification and development are distinct phases


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Other Software Process Models

- Incremental development:
  - May be plan-driven or agile (advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages flexible responses to change).
  - Specification, development and validation are **interleaved**.
- Integration and configuration:
  - May be plan-driven or agile.
  - **The system is assembled from existing configurable components**.
- In practice, most large systems use elements from each of these models.

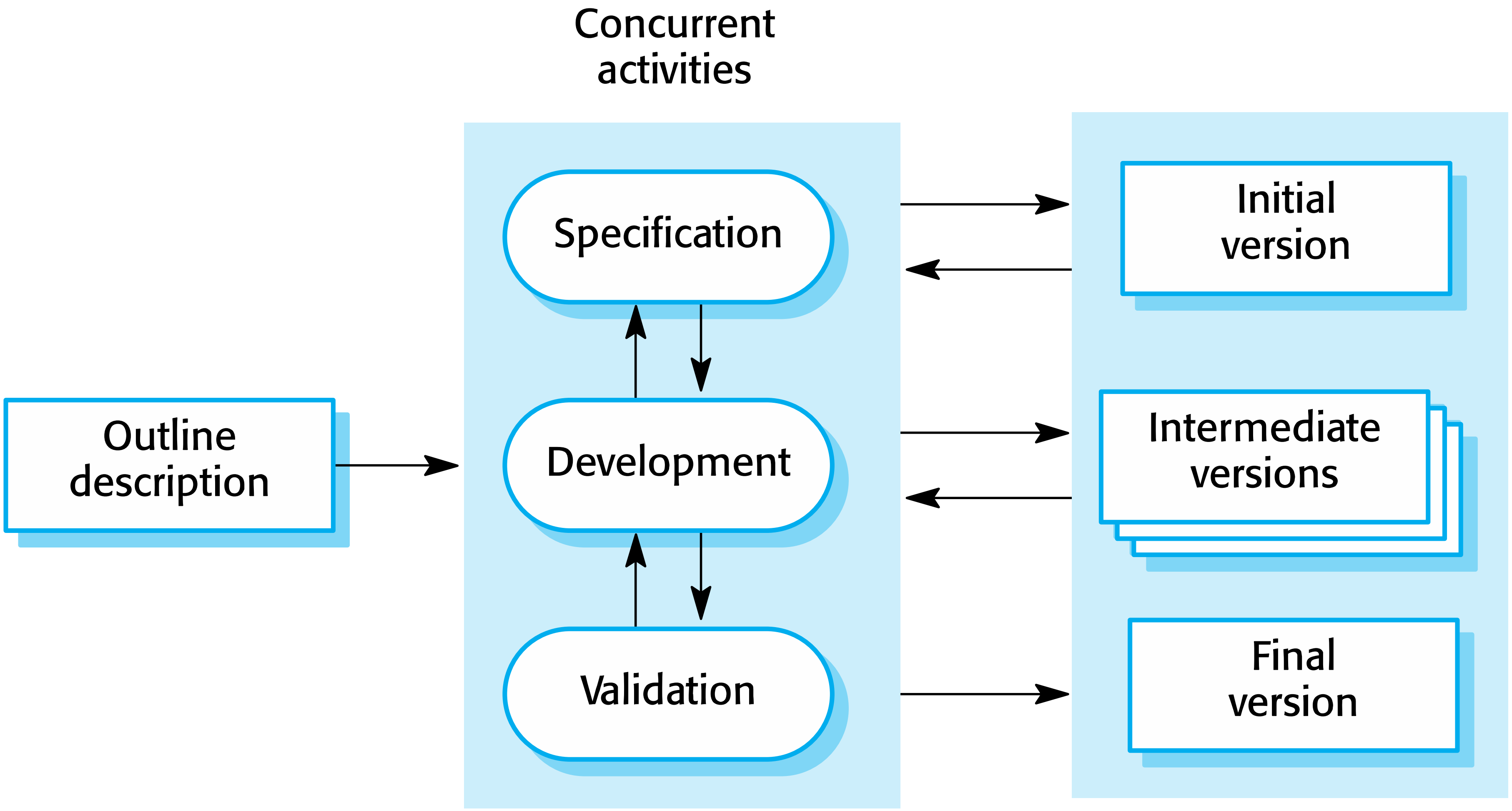
Waterfall Model

- Separate phases in the waterfall model
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Drawbacks of waterfall model
  - **Difficulty in accommodating change**
    - **In general, a phase must be complete before moving on to next phase**

Waterfall Model Properties

- Inflexibility limits its use in business systems where requirements change frequently
- Best for large systems developed over multiple sites
  - Plan driven nature helps coordinate development

Incremental Development



Incremental Development Benefits

- The cost of accommodating changing customer requirements is reduced.
  - Less specification/design for project
  - Rework of analysis/documentation is minimized.
- Easier to get customer feedback on completed development.
  - Customers can comment on demonstrations of the software
  - Customers can see how much has been implemented.
- Very rapid delivery/deployment of useful software to the customer.
  - Customers are able to use and gain value from the software quicker

Incremental Development Drawbacks

- Process is not visible
  - Managers need regular deliverables to measure progress
  - Rapid development makes it non-cost-effective to maintain documentation for all system versions
- System structure degrades with new increments
  - Extra time and money needed for refactoring
  - Alternative:
    - Regular change corrupts structure
    - Future changes become increasingly difficult and costly

Integration and Configuration

- **Based on software reuse:**
  - Systems are integrated from existing components or application systems
    - These components are sometimes called COTS (Commercial-off-the-shelf) systems
- Components may be configured to adapt behaviour and functionality to user requirements
- '***Reuse***' is now the standard approach for building many types of business system

Requirements Engineering

- Establishing:
  - Services that a customer requires from a system
  - Constraints under which it operates and is developed
  - Precise definition of behaviors which the system should exhibit
- System requirements are
  - Precise descriptions of the system services and constraints generated during requirements engineering process

Types of Requirements

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints.
  - Written primarily for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints.
  - Defines what should be implemented so may be part of a contract between client and contractor.
  - Written primarily for engineers.

Developing Requirements

- Steps:
  - Requirements elicitation: researching and discovering the requirements of a system from users, customers, and other stakeholders
  - Requirements specification: writing the formal requirements specification document
  - Requirements validation: check the requirements document for consistency, completeness and correctness
  - Requirements change:
    - inevitable changes of the specification document due to changes in user requirements, increased understanding of the stakeholders' needs, customer organizational re-structure, and availability of new technologies

Guidelines for Writing Requirements

- Choose a standard format and use it for all requirements.
- Use language in a consistent way
  - Use "***shall***" for mandatory requirements
  - Use "***should***" for desirable behaviours
    - Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon
- Include an explanation (rationale) of why a requirement is necessary

Functional and Non-functional Requirements

- Functional requirements
  - Statements of services the system should provide
  - How the system should react to particular inputs
  - How the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system
    - Timing constraints
    - Constraints on the development process
    - Standards
  - Often apply to the system as a whole rather than individual features or services
- Domain requirements
  - Constraints on the system from the domain of operation

Functional Requirements

- Describe functionality or system services.
  - Depend on the type of software, expected users and the type of system where the software is used.
  - Functional user requirements may be high-level statements of what the system should do.
  - Functional system requirements should describe the system services in detail.

Requirements Completeness and Consistency

- In principle, requirements should be both complete and consistent.
  - Complete: they should include descriptions of all facilities required
  - Consistent: there should be no conflicts or contradictions in the descriptions of the system facilities
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document

Writing Good Requirements

- Requirements must be:
  - Non-ambiguous
  - State only 1 responsibility each
  - Be Testable (i.e., verifiable)
  - Be positively stated (They should indicate what the system must do rather than what it must not do)
- Large real systems have thousands of requirements

Design – [System Architecture] and Detailed Design

- Design Process Stages:
  - System Architecture
    - Define context and modes of use of the system
    - Design system architecture [subsystems and interfaces]
  - Detailed Design
    - Identify principal system objects
    - Develop design models
    - Specify object interfaces

System Context and Interactions

- Understanding relationships between the software being designed and external environment is essential:
  - Helps decide how to provide the required system functionality
  - Helps decide how to structure system to communicate with its environment
- Understanding the context also helps establish boundaries of the system
  - Setting system boundaries helps you decide what features are implemented in the system and what features are in other associated systems

Context and Interaction Models

- System context model ➔ structural model demonstrating other subsystems in environment of the system being developed
  - Focuses on looking at your entire system and other systems around it with which it interacts
  - This may be illustrated using UML class diagrams or module diagrams
  - It is a static view of the system
- Interaction model ➔ dynamic model that shows how system interacts with its environment as it is used
  - This may be illustrated using UML sequence diagrams

Architectural Design

- Once interactions between system and environment are understood, information is used for designing system architecture
- Architectural Design: the idea is that the system will be composed of subsystems (or components).
  - Identify major components that make up system and their interactions
  - Then organize the components using an architectural pattern like layered or client-server model

Detailed Design

- Object Class Identification
- Design Models
- Subsystem Models

Object Class Identification

- Identifying object classes is often a difficult part of object oriented design
  - No 'magic formula' for object identification
  - Relies on skill, experience and domain knowledge of system designers
- Object identification is iterative. (Unlikely to get it right first time)

Approaches to Identification

- Use a grammatical approach based on a natural language description of the system
  - Base the identification on tangible things in the application domain
  - Use a behavioural approach and identify objects based on what participates in what behaviour.
  - Use a scenario-based analysis
    - The objects, attributes and methods in each scenario are identified

Design Models

- Design models show the objects/object classes and relationships between these entities
- Two kinds of design model:
  - **Structural models →** the static structure of the system in terms of object classes and relationships
  - **Dynamic models →** the dynamic interactions between objects

Examples of Design Models

- Subsystem models ➔ show logical groupings of objects into coherent subsystems
- Sequence models ➔ show the sequence of object interactions
- State machine models ➔ show how individual objects change state in response to events
- Other models ➔use-case models, aggregation models, generalisation models, etc.

Subsystem Models

- Shows how the design is organized into logically related groups of objects
- In the UML, these are shown using packages
  - An encapsulation construct - This is a logical model
  - Actual organization of objects in system may be different

Sequence Models

- Sequence models show sequence of object interactions that take place
  - Objects are arranged horizontally across the top
  - Time represented vertically so models are read top to bottom
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction
  - Thin rectangle in an object lifeline represents the time when the object is controlling object in the system

State Diagrams

- State diagrams → show how objects respond to different service requests and state transitions triggered by these requests
- State diagrams → useful high-level models of a system or an object's run-time behavior
- Don't usually need a state diagram for all objects in system
  - Many objects in system are relatively simple
  - State model adds unnecessary detail to design

Implementation [Coding]

- **Configuration management**: General process of managing a changing software system.
- Aim of configuration management is to
  - Support system integration process so all developers can access the project code and documents in a controlled way
  - All developers can find out what changes have been made
  - All developers can compile and link components to create a system

Configuration Management Activities

- **Version management:** Keep track of the different versions of software components
  - Include facilities to coordinate development by several programmers
- **System integration:** Help developers define what versions of components are used to create each version of a system
  - Description used to build system automatically by compiling and linking required components
- **Problem tracking**: Allows users to report bugs and other problems
  - Also, allow all developers see who is working on problems and when they are fixed

Development Platform Tools

- Integrated compiler/syntax-directed editing system allowing code creation, editing, and compilation
- A language debugging system.
- Graphical editing tools (i.e. edit UML models)
- Test tools (i.e. JUnit)
  - ➔ Automatically run a set of tests on a new version of a program
- Project support tools
  - ➔ Help organize code for different development projects

Integrated Development Environments (IDE)

- Software development tools often grouped to create an integrated development environment (IDE)
  - Set of software tools supporting different aspects of software development
  - Created to support development in a specific programming language such as Java
    - Language IDE may be developed specially
    - May be an instantiation of a general-purpose IDE, with specific language-support tools

Validation [Testing, Unit Test, System Test]

- Program testing is intended to show
  - a program does what it is intended to do
  - program defects before it is put into use.
- Software testing:
  - Program executed with artificial data
  - Results of the test run are checked for errors, anomalies or information about the program's non-functional attributes
  - Can reveal the presence of errors NOT their absence
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Verification vs validation

- Verification:

  "Are we building the product right”.

  - The software should conform to its specification.

- Validation:

  "Are we building the right product”.

  - The software should do what the user really requires.

Stages of testing

- **Development testing** - System is tested during development to discover bugs and defects [Unit and integration testing]
- **Release testing -** separate test team tests a complete version of the system before it is released to users [Full Qualification Testing]
    - validate each requirement (out of thousands of requirements)

Development testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - **Unit testing** - individual program units or object classes are tested
    - Unit testing focuses on testing the functionality of objects or methods
  - **Component testing** - several individual units are integrated to create composite components [a kind of Integration testing]
    - Component testing should focus on testing component interfaces
      - Send input to the component and see what comes out
  - **System testing** - All of the components in a system are integrated and the system is tested as a whole

Unit testing

- Unit testing is the process of testing individual components in isolation
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods

Release testing

- Release testing - Process of testing a release of a system intended for use outside the development team
- Primary goal is to convince the supplier of the system that it is good enough for use
  - In the end, Release testing has to show:
    - System delivers its specified functionality, performance and dependability
    - System does not fail during normal use
- Release testing usually a black-box testing process where tests are only derived from the system specification [Requirements based testing]

Release testing and system testing

- Release testing is a form of system testing
- Important differences:
  - A separate team not involved in system development, is responsible for release testing
  - System testing by development team should focus on discovering bugs in the system (defect testing)
  - Objective of release testing is to check that system meets its requirements and is good enough for external use (validation testing)

Deployment

- This stage may involve:
  - Dry runs with a reduced system but real user data
    - give real user data and check if the system works fine
  - Full deployment

Evolution [Maintenance]

- Changes may be required by user after deployment
  - New requirements/modified requirements
  - Fix bugs/deficiencies not caught in testing
- Process should be organized so changes can be traced
- Generally, design process assures there are links between
  - Requirements
  - Architecture/design
  - Test cases/procedures
- Documentation must be maintained during evolution

Evolution

- Typical process:
  - Change/update proposed by user or systems staff. Proposal includes
    - Specific deficiency or information on new requirement
    - Rationale
    - Other info as needed
  - A Change Control Board (CCB) reviews request and responds
    - Accepted [Assign persons responsible for change]
    - Rejected [Reason for rejection]
    - Request for Info [Request for additional data for clarification

Evolution

- Once approved:
  - Requirements are updated and reviewed
  - Design modified/reviewed (links to requirements updated as needed)
  - Implementation written/code modified
  - New code tested
  - Possible regression testing
  - Changes are accepted and system is updated in source and documentation versioning