Introduction to Software Engineering

Gérald Monard

Ecole GDR CORREL - April 16, 2013

www.monard.info

Bibliography

- *Software Engineering, 9th ed.*
  (I. Sommerville, 2010, Pearson)
- *Conduite de projets informatiques, 2nde éd.*
  (B.-A. Guérin, 2012, ENI)
- *Computer Science, An Overview, 10th ed.*
  (J. G. Brookshear, 2010, Pearson)
- *Design Patterns Explained*
  (A. Shalloway, J. R. Trott, 2002, Addison-Wesley)
- *Design Patterns, Elements of Reusable Object-Oriented Software*
  (E. Gamma, R. Helm, R. Johnson, J. Vlissides, 1995, Addison-Wesley)
- *Sam's Teach Yourself UML in 24 hours*
  (J. Schmuller, 2006, Sams publishing)

Introduction

Why this course?

Software: Set of computer programs and associated documentation

Personal software vs. Professional Software: When one writes a program for itself, he/she will be the only user, and he/she does not need to care about writing program guides, documenting the program design,. . . Professional software is developed by teams rather than individuals, and is intended to be used by other people than the developers.

Software Engineering is intended to support professional software development rather that individual programming. It includes techniques that support program specification, design, and evolution.

This course targets an introduction to the main aspects of software engineering. It will show you that a “good” software is not only a few lines of codes...

Introduction

What is a "good" software?

Good software should deliver the required functionality and performance to the user and should be:

- maintainable: it should be written in such a way so that it can evolve to meet the changing needs of users
- dependable: it should be reliable, secure, and safe
- usable: it should be acceptable to the type of users for which it is designed
- efficient: it should not make wasteful use of system resources such as memory and processor cycles

Introduction

What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production, from the early stages of system specification through to maintaining the system after it has gone into use.

What are the fundamental software engineering activities?

- Software specification
- Software development
- Software validation
- Software evolution

What is the difference between software engineering and computer science?

Computer science focuses on theory and fundamentals. Software engineering is concerned with the practicalities of developing and delivering useful software

Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

Software Processes

What is a software process?

- ➢ A software process is a set of related activities that leads to the production of a software product.
- ➢ These activities may involve:
  - ✓ the development of software from scratch
  - ✓ the extensions and modifications of existing systems
  - ✓ the configuration and integration of off-the-shelf software or system components

Software Processes

There are many different software processes, but all include:

Software specification

The functionality of the software and constraints on its operation must be defined

Software design and implementation

The software to meet the specification must be produced

Software validation

The software must be validated to ensure that it does what the customer wants

Software evolution

The software must evolve to meet changing customer needs

Software process models

Software processes are complex and there is no ideal process. Most organizations have developed their own software development processes. Most software processes can be categorized as either:

- plan-driven processes, where all of the process activities are planned in advance and progress is measured against this plan.
- agile processes, where planning is incremental and it is easier to change the process to reflect changing customer requirements.

Generally, organizations try to find a balance between plan-driven and agile processes

Software process models

Some generic process models:

- waterfall model This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases
- incremental development This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version
- reuse-oriented software engineering This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch

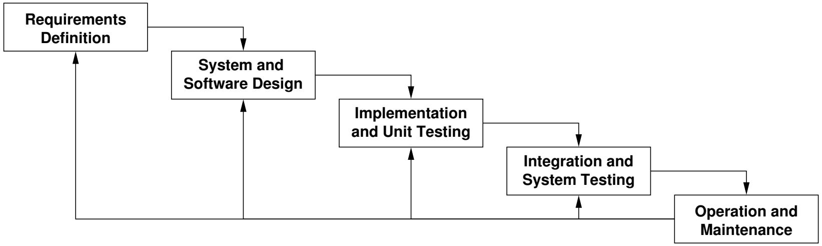
These models are not exclusive and are often used together. For example:

- Parts well understood ☞ waterfall-model process
- Parts that are difficult to specify in advance (e.g., user interface) ☞ incremental approach

Software process models

The waterfall model

- The waterfall model is an example of a plan-driven process. In principle, you must plan and schedule all of the process activities before starting to work on them.
- It is the first published model of software development process (1970).


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Software process models

The waterfall model

The principal stages of the waterfall model directly reflect the fundamental development activities:

- Requirements analysis and definition The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification
- System and software design The system design process establishes an overall system architecture, while software design involves identifying and describing the fundamental software system abstractions and their relationships
- Implementation and unit testing During this stage, the software design is realized with a set of programs or program units. Unit testing involves verifying that each unit meets its specification

Software process models

The waterfall model

- Integration and system testing The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer
- Operation and maintenance This is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered

In principle, the result of each phase is one or more documents that are approved ('signed off'). The following phase should not start until the previous phase has finished. In practice, these stages overlap and feed information to each other.

Software process models

The waterfall models: pros ans cons

- It is consistent with other engineering process models and documentation is produced at each phase. This makes the process visible so managers can monitor progress against the development plan.
- Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.

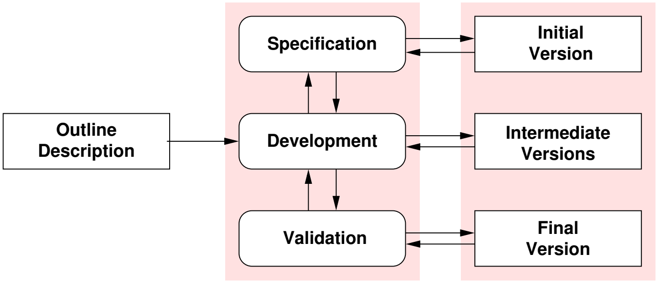
When to use the waterfall model

- In principle, the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.
- a waterfall model use case: formal system development (the system specification is a mathematical model)

Software process models

Incremental development

- Incremental development is based in the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed.
- Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities



Software process models

Incremental development

- Incremental software development is usually better than a waterfall approach
- Incremental development reflects the way that we solve problems
- Each increment or version of the system incorporate some of the functionality that is needed by the customer.
- Generally, the early increments of the system include the most important or the most urgently required functionality. This gives the user the possibility of evaluating the system at a relatively early stage in the development.

Software process models

Incremental development: pros and cons

Incremental development has three important benefits, compared to the waterfall model:

- The cost of accommodating changing customer requirements is reduced (amount of analysis and documentation that has to be redone)
- It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Software process models

Incremental development: pros and cons

From a management perspective, the incremental approach has two problems

1. The process is not visible. Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system
2. System structure tends to degrade as new increments are added. Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

Today, incremental development in some form is the most common approach for the development of application systems. But it is not well-adapted to large, complex, long-lifetime systems, where different teams develop different parts of the system. Large systems need a stable framework that has to be planned in advance.

Software process models

The goal of software processes is to specify, design, implement, and test a software system.

Software specification

- Software specification (or requirements engineering) is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
- This is a critical stage of the software process since errors at this stage inevitably lead to later problems in the system design and implementation.

Software process models

Software specification

- There are four main activities in this process:
  1. *Feasibility study*
  2. *Requirements elicitation and analysis*
     This is the process of requirements through observation of existing systems, discussion with potential users, task analysis, etc. These help understanding the system to be specified
  3. *Requirements specification*
     This is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements (user and system requirements)
  4. *Requirements validation*
     This activity checks the requirements for realism, consistency, and completeness

Software process models

Software design and implementation

- The implementation stage of software development is the process of converting a system specification into an executable system.
- It always involves processes of software design and programming, with sometimes a refinement of software specification (incremental approach)
- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.
- The process of software design is often developed iteratively.

Software process models

Software validation

- Software validation is intended to show that a system both conforms to its specification and that it meets the expectations of the system customer.
- Program testing, where the system is executed using simulated test data, is the principal validation technique
- Except for small programs, systems should not be tested as a single monolithic unit.

Software process models

Software validation

- The stages in the testing process are:
  1. Development testing The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components.
  2. System testing System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.
  3. Acceptance testing The system is tested with data supplied by the customer rather that with simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition.

Software process models

Software evolution (maintenance)

- Software development does not stop when a system is delivered
- It continues throughout the lifetime of the system
- The costs of maintenance are often several times the initial development costs
- Hardly any software systems are completely new systems and it makes much more sense to see development and maintenance as a continuum.
- Software engineering can be seen as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

Requirements engineering (software requirements)

Introduction

- The requirements for a system are the descriptions of what the system should do: the services that it provides and the constraints on its operation.
- The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering.
- Requirements can be divided into:
  1. User requirements These are statements of what services the system is expected to provide to system users and the constraints under which it must operate
  2. System requirements There are more detailed description of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented.

Requirements engineering

Functional and non-functional requirements

Software system requirements are often classified as:

- Functional requirements These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. They can also include explicitly what the system should not do.
- Non-functional requirements These are constraints on the services or functions offered by the system, They include timing constraints, constraints on the development process, and constraints imposed by standards.

Requirements engineering

Functional requirements

- The functional requirements for a system describe what the system should do
- They are usually described in an abstract way that can be understood by system users.
- In principle, the functional requirements specification of a system should be both complete and consistent.
  - ✓ Completness means that all services required by the user should be defined
  - ✓ Consistency means that requirements should not have contradictory definitions
- In practice, for large, complex systems, it is practically impossible to achieve requirements consistency and completeness

Requirements engineering

Non-functional requirements

- Non-functional requirements are requirements that are not directly concerned with the specific services delivered by the system to its users.
- Non-functional requirements, such as performance, security, or availability, usually specify or contrain characteristics of the system as a whole
- Non-functional requirements are often more critical than individual functional requirements: failing to meet a non-functional requirement can mean that the whole system is unusable.
- Non-functional requirements can relate for example about speed, size, ease of use, reliability, robustness, portability, etc.

Requirements engineering

The software requirements document

- The software requirements document(/specification) is an official statement of what the system developers should implement.
- It should include both the user and the system requirements
- The level of details included in the document depends on the type of system that is being developed and the development process used (plan-driven vs. incremental).

Requirements specification

- The requirement specification is the process of writing down the user and system requirements in a requirements document.
- Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent.

Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

System modeling

What is system modeling?

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling generally means representing the system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML)
- The most important aspect of a system model is that it leaves out detail
- A model is an abstraction of the system being studied rather than an alternative representation of that system.

System modeling

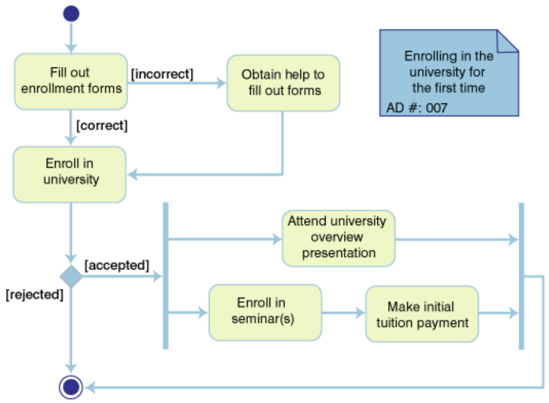
The Unified Modeling Language (UML)

- UML is a set of 13 different diagram types that may be used to model software systems.
- It emerges in 1990s on object-oriented modeling
- A major revision (UML2) was finalized in 2004
- Most users of UML use five diagram types to represent the essentials of a system:
  1. Activity diagrams, which show the activities involved in a process or in data processing
  2. Use case diagrams, which show the interactions between a system and its environment
  3. Sequence diagrams, which show interactions between actors and the system and between system components
  4. Class diagrams, which show the object classes in the system and the associations between these classes
  5. State diagrams, which show how the system reacts to internal and external events

System modeling

Activity diagrams

- Activity diagrams are intended to show the activities that make up a system process and the flow of control from one activity to another.

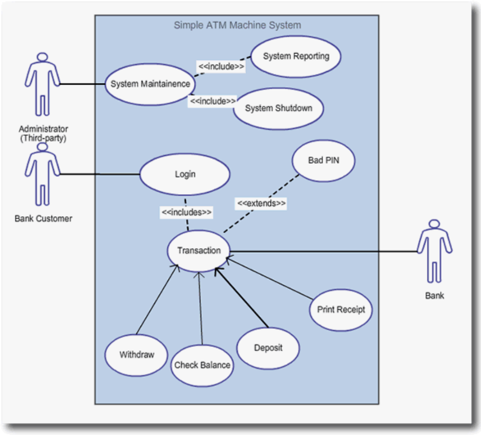


Fill out enrollment forms
[incorrect]
Obtain help to fill out forms
Enrolling in the university for the first time
AD #: 007
[correct]
Enroll in university
Attend university overview presentation
[accepted]
[rejected]
Enroll in seminar(s)
Make initial tuition payment

SYSTEM MODELING

Use case diagrams

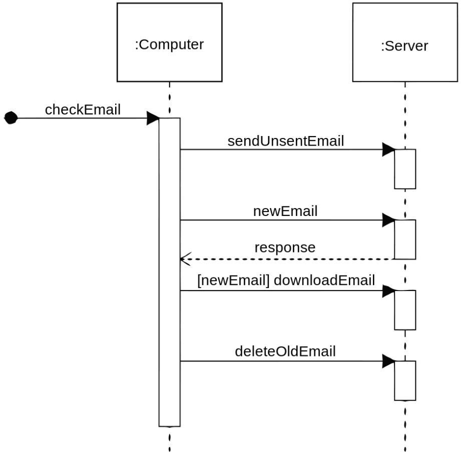
- ➢ A use case can be taken as a simple scenario that describes what a use expects from a system
- ➢ Each use case represents a discrete task that involves external interaction with a system



System modeling

Sequence diagram

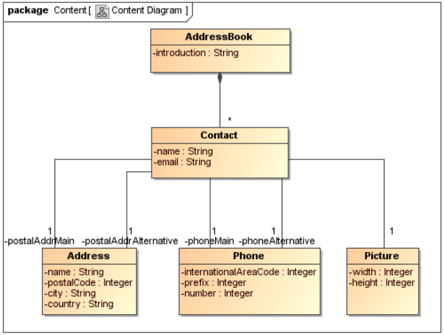
- ➢ Sequence diagrams are primarily used to model the interactions between the actors and the objects in a system and the interactions between the objects themselves.
- ➢ A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.



System modeling

Class diagram

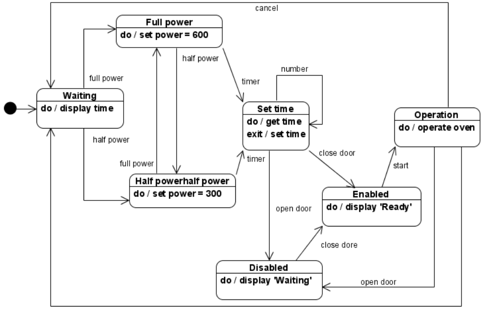
- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes
- An association is a link between classes that indicates that there is a relationship between these classes (composition, inheritance, etc.)



System modeling

State diagram

- ➢ State diagrams show system states and events that cause transitions from one state to another
- ➢ They do not show the flow of data within the system but may include additional information on the computations carried out in each state



Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

Architectural Design and Implementation

Architectural design

- Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system.
- The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.
- In the incremental model, it is generally accepted that an early stage of the development process should be concerned with establishing an overall system architecture.
- However, incremental development of architectures is not usually successful. Refactoring a system architecture is likely to be expensive.
- Ideally, a system specification should not include any design information. In practice, this is untrue. Architectural decomposition is usually necessary to structure and organize the specification.

Architectural Design and Implementation

Architectural design

- Software architecture is important because it affects the performance, robustness, distributability, and maintainability of a system.
- Individual components implement the functional system requirements.
- The non-functional requirements depend on the system architecture.
- In many systems, non-functional requirements are also influenced by individual components, but the architecture of the system is the dominant influence.

Architectural Design and Implementation

Architectural patterns

- The architecture of a software system may be based on a particular architectural pattern or style.
- An architectural pattern is a description of a system organization, such as a client-server organization or a layered architecture, etc.
- Architectural patterns capture the essence of an architecture that has been used in different software systems.
- Commonly used architectural patterns include:
    - ✓ Model-View-Controller
    - ✓ Layered Architecture
    - ✓ Repository
    - ✓ Client-server
    - ✓ Pipe and filter

Architectural Design and Implementation

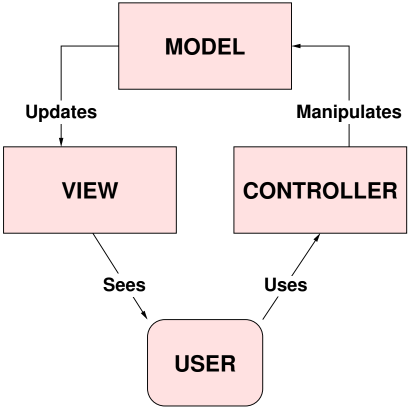
The Model-View-Controller pattern

- ➢ The MVC software architecture pattern separates presentation and interaction from the system data.
- ➢ The system is structured in to three logical components that interact with each other.
  - ✓ The Model component manages the system data and associated operations on that data.
  - ✓ The View component defines and manages how the data is presented to the user.
  - ✓ The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes there interactions to the View and the Model components

Architectural Design and Implementation

The Model-View-Controller pattern

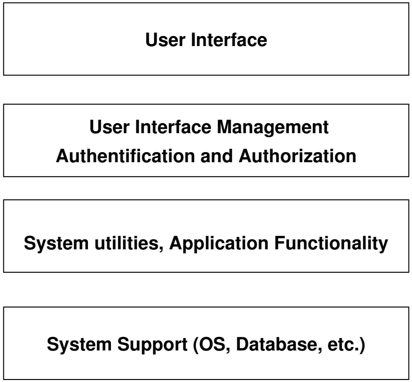
- ➢ The MVC pattern is used when there are multiple ways to view and interact with the data
- ➢ It allows the data to change independently of its representation and vice-versa.
- ➢ Examples of MVC patterns: many websites, graphical interfaces to program (e.g., GaussView/Gaussian), etc.



Architectural Design and Implementation

The Layered architecture pattern

- The layered architecture is another of achieving separation and independence
- The system functionality is organized into separate layers, and each layer only relies on the facilities and services offered by the layer immediately beneath it


User Interface
User Interface Management
Authentification and Authorization
System utilities, Application Functionality
System Support (OS, Database, etc.)

Architectural Design and Implementation

The Layered architecture pattern

- The layered architecture allows incremental approach: as a layer is developed, some of the services provide by that layer may be made available to the others
- It is also changeable and portable: so long as its interface is unchanged, a layer can be replaced by another, equivalent layer. Furthermore, when layer interfaces change or new facilities are added to a layer, only the adjacent layer is affected
- It makes also easier to provide multi-platform implementations by localizing machine dependencies in inner layers

Architectural Design and Implementation

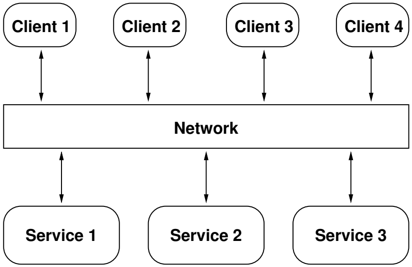
The Client-Server architecture pattern

- In a client-server architecture, the functionality of the system is organized into services, with each service delivered from a separate server.
- Clients are users of these services and access servers to make use of them.
- The major components of this model are:
  1. A set of servers that offer services to other components
  2. A set of clients that call on the services offered by servers. This usually means several instances of a client program executing concurrently on different computers.
  3. A network that allows the clients to access these services.

ARCHITECTURAL DESIGN AND IMPLEMENTATION

The Client-Server architecture pattern

- An important benefit is separation and independence. Services and servers can be changed without affecting other parts of the system.
- Clients access the services provided by a server through remote procedure calls using a request-reply protocol
- For example, in the http protocol, a client makes a request to a server and waits until it receives a reply.



Architectural Design and Implementation

Implementation issues

- A critical stage of software development is, of course, system implementation, where one creates an executable version of the software.
- We don't cover here "good" programming practices that are usually language specific.
- However, there are some aspects of implementation that are particularly important and that are language-independent:
  1. Reuse Most modern software is constructed by reusing existing components or systems. You should make as much use as possible of existing codes
  2. Configuration Management During the development process, many different versions of each software component are created. You should keep track of them
  3. Host-target development Production software does not usually execute on the same computer as the software development environment. You should separate the host system (where you develop) and the target system (where you execute).

Architectural Design and Implementation

Implementation issues: Reuse

- By reusing existing software, you can develop new systems more quickly, with fewer development risks and also lower costs.
- As the reused software has been tested in other applications, it should be more reliable than new software.
- However, there are costs associated with reuse:
  1. The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs
  2. The costs of buying the reusable software (where applicable)
  3. The costs of adapting and configuring the reusable software components or systems
  4. The costs of integrating reusable software elements with each other and with the new code that you have developed

Architectural Design and Implementation

Implementation issues: Configuration Management

- In software development, change happens all the time, so change management is essential
- You have to ensure that:
  - ✓ team members don't interfere with each other's work
  - ✓ changes on the same component are coordinated
  - ✓ everyone can access the most up-to-date versions of software components
  - ✓ when something goes wrong with a new version, you have to be able to go back to a working version of the system or component
- Configuration management is the name given to the general process of managing a changing software system
- The aim of configuration management is to support the system integration process so that developers :
  - ✓ can access the project code and documents in a controlled way
  - ✓ find out what changes have been made
  - ✓ compile and link components to create a system

Architectural Design and Implementation

Implementation issues: Configuration Management

There are three fundamental configuration management activities:

- Version management where support is provided to keep track of the different versions of software components
- System integration where support is provided to help developers define what versions of components are used to create each version of a system
- Problem tracking where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed

Architectural Design and Implementation

Implementation issues: Host-target development

- Most software development is based on a host-target model
- The software is developed on a development platform
- It runs on an execution platform
- A platform is more than just hardware. It includes the installed OS, plus other supporting software such as a database management system, or an interactive development environment (IDE)
- If the targeted execution platform is not the development platform, it is necessary either to move the development software to the execution platform for testing or run a simulator on the development machine

Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

Software testing

Introduction

- ➢ Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use
- ➢ The testing process has two distinct goals:
  1. To demonstrate to the developer and the customer that the software meets its requirements. This is called validation testing.
     - ☞ For custom software, this means that there should be at least one test for every requirement in the requirements document
     - ☞ For generic software products, it means that there should be tests for all of the system features, plus combinations of these features
  2. To discover situations in which the behavior of the software is incorrect, undesirable, or does not conform to its specification. This is called defect testing.
- ➢ Testing cannot demonstrate that the software is free of defects or that it will behave as specified in every circumstance.

Testing can only show the presence of errors, not their absence
E. Dijkstra said (1972)

Software testing

Introduction

- ➢ Testing is part of a broader process of software verification and validation
  - ✓ Validation: Are we building the right product?
  - ✓ Verification: Are we building the product right?
- ➢ The ultimate goal of verification and validation processes it to establish confidence that the software system is “fit for purpose”.
- ➢ The verification and validation process may involve software inspections and reviews.
- ➢ Software inspections and reviews analyze and check the system requirements, design models, the program source code, and even proposed system test.
- ➢ Software inspections and reviews like automated static analysis of the source code are related to quality management

Software testing

Development testing

- Development testing includes all testing activities that are carried out by the team developing the system.
- During development, testing may be carried out at three levels of granularity
  1. Unit testing, where individual program units or object classes are tested. Unit testing focuses o testing the functionality of objects or methods.
  2. Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces
  3. System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions

Outline

1. Software Processes
   what are the activities involved in producing a software?
2. Requirement Engineering
   How to define what a software should (and should not) do?
3. System Modeling
   How to develop abstract models of a system, with each model presenting a different view or perspective of that system?
4. Architectural Design and Implementation
   How a software should be organized?
5. Software testing
   What kind of tests should be carried out to validate a software?
6. Software evolution
   How to help your software evolving?

Software evolution

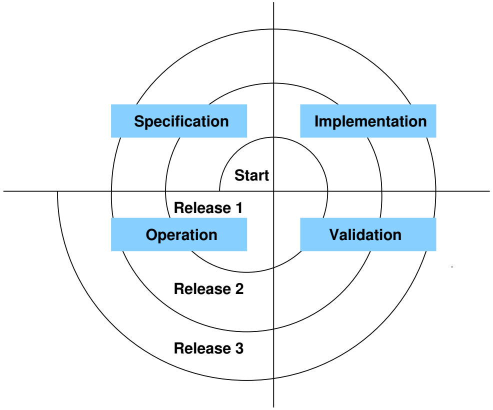
Introduction

- Software development does not stop when a system is delivered but continues throughout the lifetime of the system.
- After a system has been deployed, it inevitably has to change if it is to remain useful.
- Business changes and changes to user expectations generate new requirements
- Parts of the software may have to be modified to correct errors that are found inoperation, to adapt it for changes to its hardware and software platform, and to improve its performance or other non-functional characteristics
- Useful software systems often have very long lifetime!

Software evolution

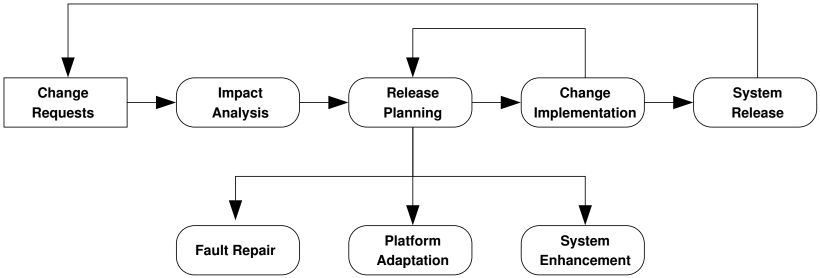
Introduction

- Software engineering can be seen as a spiral process with requirements, design, implementation, and testing going on throughout the lifetime of the system



Software evolution

Overview of the evolution process



- During release planning, all proposed changes are considered (fault repair, adaptation, and new functionality)
- Change implementation can be seen as an iteration of the development process, where the revisions to the system are designed, implemented, and tested.