Chapter1. The Software Life Cycle

What is software?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled.
- Expenditure on software represents a significant fraction of GNP in all developed countries.
- More and more individuals and society rely on advanced software systems.

- Software is computer programs and associated documentation such as requirements, design models and user manuals.

- Software products may be developed for a particular customer or may be developed for a general market.
- Software products may be
  - Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - Bespoke (custom) - developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

What is good software?

- Good software should deliver the required functionality and performance to the user and should be maintaintable, dependable, and usable.

- Essential attributes for good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the types of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

What is software engineering?

- Software engineering is an engineering discipline that is concerned with all aspects of software production from early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- All aspects of software production
  - Not just technical process of development. Also project management and the development of tools, methods, etc to support software production.

- Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

Importance of software engineering

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.

- Software engineering is concerned with cost-effective software development.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs is the cost of changing the software after it has gone into use.

What is a software process?

- A set of activities whose goal is the development or evolution of software.
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands.

1.1 Introduction

- The software life cycle : the sequence of different activities that take place during software development.

TYPES OF SOFTWARE LIFE CYCLE ACTIVITIES

1. Feasibility—Determining if the proposed development is worthwhile.
   Market analysis—Determining if there is a potential market for this product.

2. Requirements—Determining what functionality the software should contain.
   - Requirement elicitation—Obtaining the requirements from the user.
   - Domain analysis—Determining what tasks and structures are common to this problem.

3. Project planning—Determining how to develop the software.
   - Cost analysis—Determining cost estimates.
   - Scheduling—Building a schedule for the development.
   - Software quality assurance—Determining activities that will help ensure quality of the product.
   - Work-breakdown structure—Determining the subtasks necessary to develop the product.

TYPES OF SOFTWARE LIFE CYCLE ACTIVITIES

4. Design—Determining how the software should provide the functionality.
   - Architectural design—Designing the structure of the system.
   - Interface design—Specifying the interfaces between the parts of the system.
   - Detailed design—Designing the algorithms for the individual parts.

5. Implementation—Building the software.

6. Testing—Executing the software with data to help ensure that the software works correctly.
   - Unit testing—Testing by the original developer.
   - Integration testing—Testing during the integration of the software.
   - System testing—Testing the software in an environment that matches the operational environment.

   - Alpha testing—Testing by the customer at the developer’s site.
   - Beta testing—Testing by the customer at the customer’s site.
   - Acceptance testing—Testing to satisfy the purchaser.
   - Regression testing—Saving tests from the previous version to ensure that the new version retains the previous capabilities.

TYPES OF SOFTWARE LIFE CYCLE ACTIVITIES

7. Delivery—Providing the customer with an effective software solution.
    - Installation—Making the software available at the customer's operational site.
    - Training—Teaching the users to use the software.
    - Help desk—Answering questions of the user.

8. Maintenance—Updating and improving the software to ensure continued usefulness.

TYPICAL DOCUMENTS

1. Statement of work—Preliminary description of desired capabilities, often produced by the user.

2. Software requirements specification—Describes what the finished software will do.
   - Object model—Shows main objects/classes.
   - Use case scenarios—Show sequences of possible behaviors from the user's viewpoint.

3. Project schedule—Describes the order of tasks and estimates of time and effort necessary.

4. Software test plan—Describes how the software will be tested to ensure proper behavior.

   Acceptance tests—Tests designated by the customer to determine acceptability of the system.

TYPICAL DOCUMENTS

5. Software design—Describes the structure of the software.
    - Architectural design—The high-level structure with the interconnections.
    - Detailed design—The design of low-level modules or objects.

6. Software quality assurance plan (SQA plan)—Describes the activities that will be done to ensure quality.

7. User manual—Describes how to use the finished software.

8. Source code—The actual product code.

9. Test report—Describes what tests were done and how the system behaved.

10. Defect report—Describes dissatisfaction of the customer with specific behavior of the system; usually, these are software failures or errors.

1.2 Software Life Cycle Models

- THE LINEAR SEQUENTIAL MODEL
  - waterfall model
  - First described by Royce in 1970
  - There are many versions of the waterfall model.
  - Note that in this version of the waterfall, the project planning activities are included in the requirements phase. Similarly, the delivery and maintenance phases have been left off.

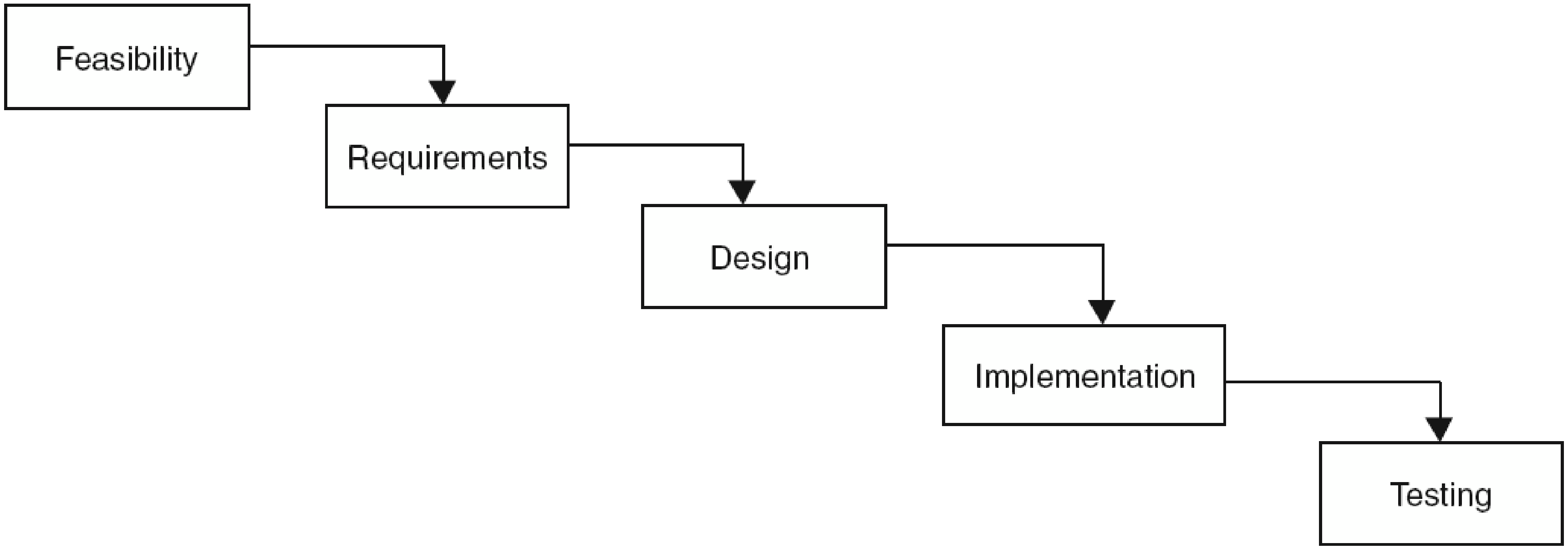


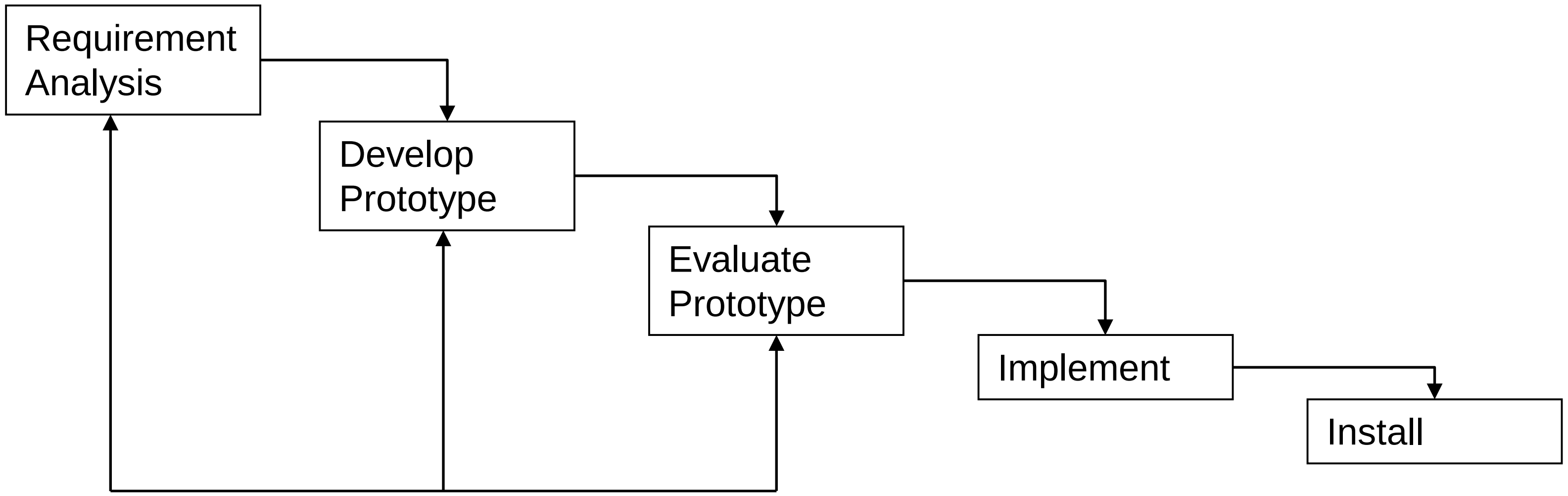
Fig. 1-1. Waterfall model.

1.2 Software Life Cycle Models

- THE LINEAR SEQUENTIAL MODEL
  - The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In practice, a phase has to he complete before moving onto the next phase.
  - Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
    - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
    - Few business systems have stable requirements.
  - The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

SOFTWARE LIFE CYCLE MODELS

- THE PROTOTYPING MODEL
  - a throwaway version (or prototype)
  - This prototype is intended to test concepts and the requirements.
  - The prototype will be used to demonstrate the proposed behavior to the customers. After agreement from the customer, then the software development usually follows the same phases as the linear sequential model.



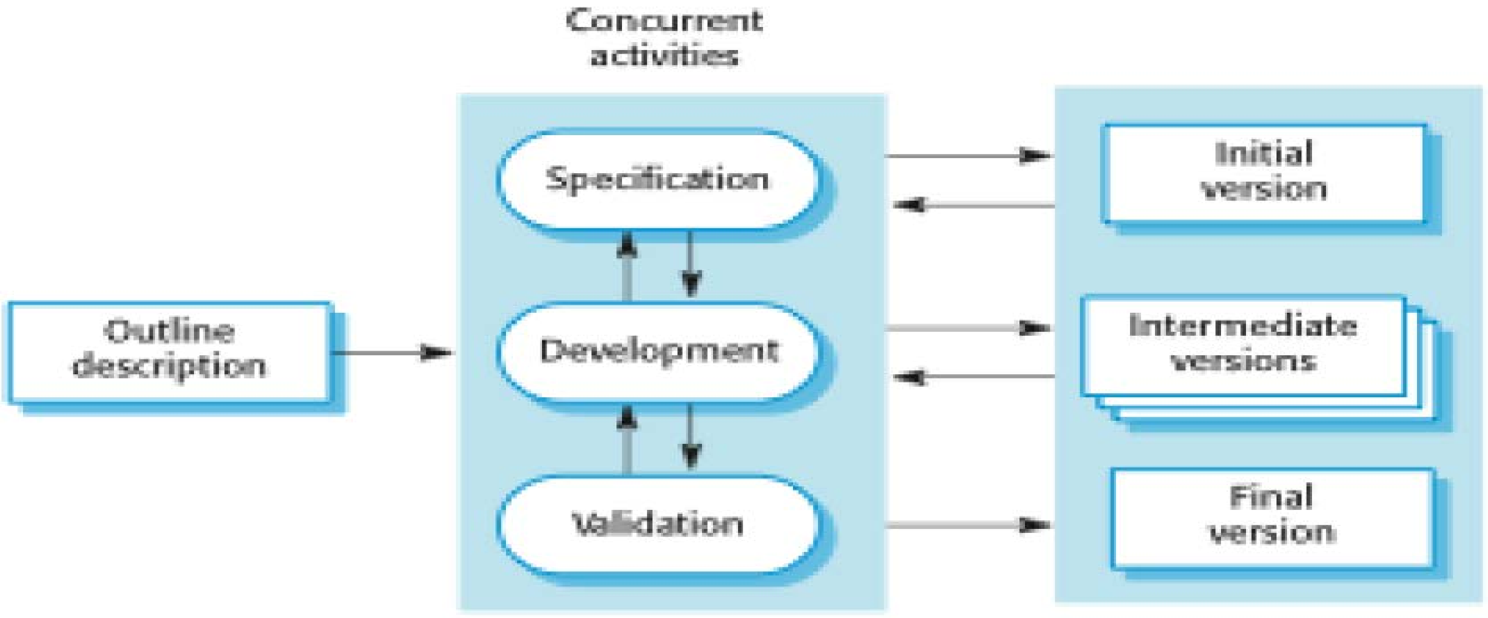
1.2 Software Life Cycle Models

- THE PROTOTYPING MODEL Benefits
  - Improved system usability
  - A closer match to users' real needs
  - Improved design quality
  - Improved maintainability
  - Reduced development effort

- THE PROTOTYPING MODEL Problems
  - Error checking and recovery may not be included in the prototype.
  - Focus on functional rather than non-functional requirements such as reliability and security.

SOFTWARE LIFE CYCLE MODELS

- INCREMENTAL MODEL
  - D. L. Parnas proposed the incremental model.
  - The goal was to design and deliver to the customer a minimal subset of the whole system that was still a useful system.
  - The process will continue to iterate through the whole life cycle with additional minimal increments. The advantages include giving the customer a working system early and working increments.



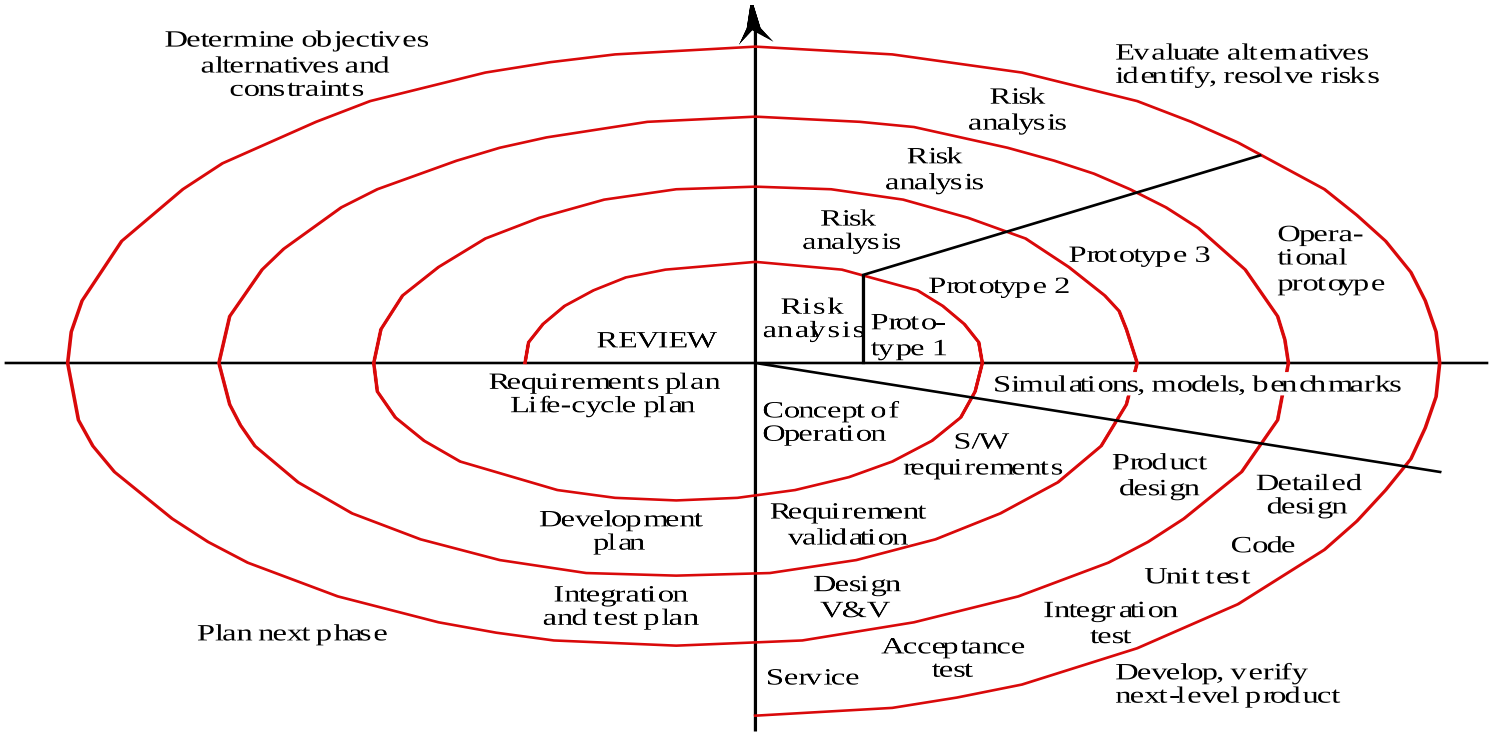
SOFTWARE LIFE CYCLE MODELS

- INCREMENTAL MODEL Benefits
  - The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
  - 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.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

- INCREMENTAL MODEL Problems
  - 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.
  - 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.

SOFTWARE LIFE CYCLE MODELS

- BOEHM'S SPIRAL MODEL
  - B. Boehm introduced the spiral model.
  - The image of the model is a spiral that starts in the middle and continually revisits the basic tasks of customer communication, planning, risk analysis, engineering, construction and release, and customer evaluation.

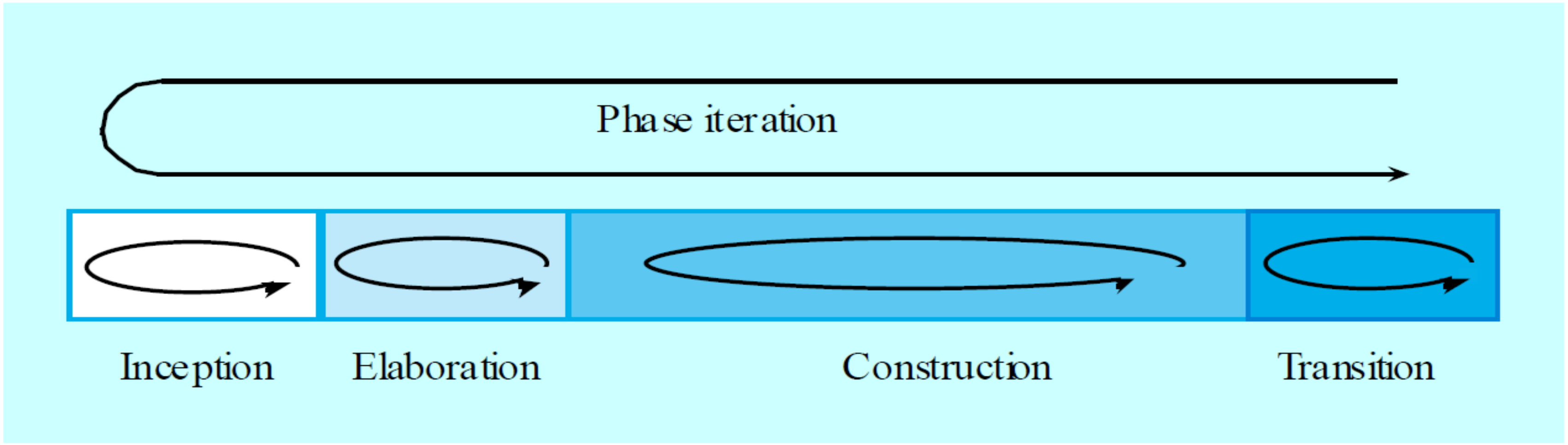


SOFTWARE LIFE CYCLE MODELS

- BOEHM’S SPIRAL MODEL
  - Process is represented as a spiral rather than as a sequence of activities with backtracking.
  - Each loop in the spiral represents a phase in the process.
  - No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
  - Risks are explicitly assessed and resolved throughout the process.
- BOEHM’S SPIRAL MODEL Sectors
  - Objective setting
    - Specific objectives for the phase are identified.
  - Risk assessment and reduction
    - Risks are assessed and activities put in place to reduce the key risks.
  - Development and validation
    - A development model for the system is chosen which can be any of the generic models.
  - Planning
    - The project is reviewed and the next phase of the spiral is planned.

SOFTWARE LIFE CYCLE MODELS

- The Rational Unified Process Model
  - RUP Phases
    - Inception
       Establish the business case for the system.
    - Elaboration
       Develop an understanding of the problem domain and the system architecture.
    - Construction
       System design, programming and testing.
    - Transition
       Deploy the system in its operating environment.



SOFTWARE LIFE CYCLE MODELS

- The Rational Unified Process Model
  - In-phase iteration
    - Each phase is iterative with results developed incrementally.
  - Cross-phase iteration
    - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.
-

SOFTWARE LIFE CYCLE MODELS

- RUP good practice
  - Develop software iteratively
    - Plan increments based on customer priorities and deliver highest priority increments first.
  - Manage requirements
    - Explicitly document customer requirements and keep track of changes to these requirements.
  - Use component-based architectures
    - Organize the system architecture as a set of reusable components.
  - Visually model software
    - Use graphical UML models to present static and dynamic views of the software.
  - Verify software quality
    - Ensure that the software meet's organizational quality standards.
  - Control changes to software
    - Manage software changes using a change management system and configuration management tools.

SOFTWARE LIFE CYCLE MODELS

- Agile Method
  - Rapid development and delivery is now often the most important requirement for software systems
    - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
    - Software has to evolve quickly to reflect changing business needs.
  - Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
    - Focus on the code rather than the design
    - Are based on an iterative approach to software development
    - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
  - The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

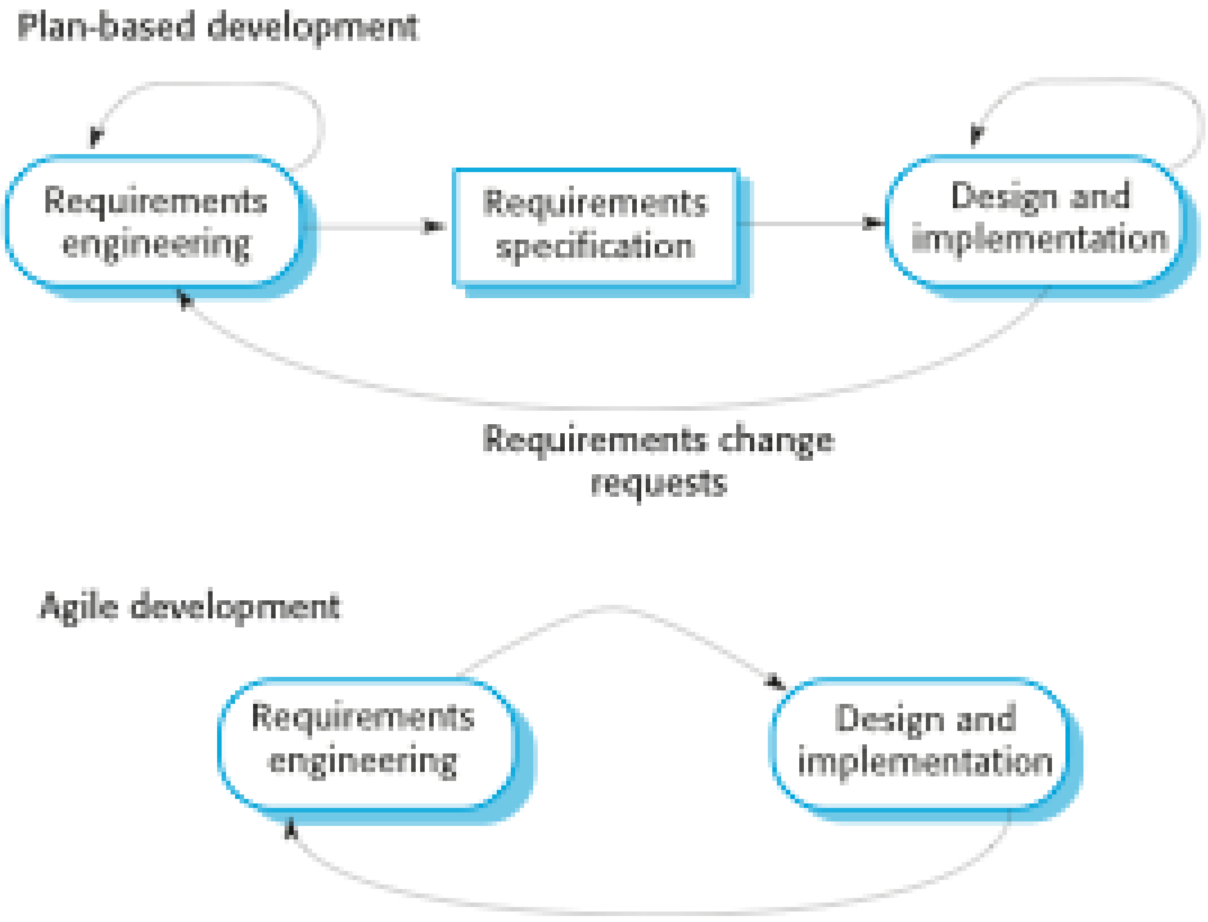
SOFTWARE LIFE CYCLE MODELS

- The Principles of Agile Method

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

SOFTWARE LIFE CYCLE MODELS

- Plan-driven development vs Agile development


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

SOFTWARE LIFE CYCLE MODELS

- Extreme programming
  - Perhaps the best-known and most widely used agile method.
  - Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
    - New versions may be built several times per day;
    - Increments are delivered to customers every 2 weeks;
    - All tests must be run for every build and the build is only accepted if tests run successfully.
  - Incremental development is supported through small, frequent system releases.
  - Customer involvement means full-time customer engagement with the team.
  - People not process through pair programming, collective ownership and a process that avoids long working hours.
  - Change supported through regular system releases.
  - Maintaining simplicity through constant refactoring of code.

SOFTWARE LIFE CYCLE MODELS

- Extreme programming practices

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

SOFTWARE LIFE CYCLE MODELS

- Extreme programming practices

Pair programming	Developers work in pairs, checking each other’s work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

SOFTWARE LIFE CYCLE MODELS

- Problems with agile method
  - It can be difficult to keep the interest of customers who are involved in the process.
  - Team members may be unsuited to the intense involvement that characterizes agile methods.
  - Prioritizing changes can be difficult where there are multiple stakeholders.
  - Maintaining simplicity requires extra work.
    - Problems may arise if original development team cannot be maintained.
  - Contracts may be a problem as with other approaches to iterative development.