MODULE 1

Introduction to Software Engineering - Professional software development, Software engineering ethics. Software process models - The waterfall model, Incremental development. Process activities - Software specification, Software design and implementation, Software validation, Software evolution. Coping with change - Prototyping, Incremental delivery, Boehm's Spiral Model. Agile software development - Agile methods, agile manifesto - values and principles. Agile development techniques, Agile Project Management. Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care.

Software

Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.

Attributes of good software

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

Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production.

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.

Fundamental software engineering activities

Software specification, software development, software validation, and software evolution.

Software engineering and system engineering

System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.

Key challenges facing software engineering

Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.

Costs of software engineering

Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs

Best software engineering techniques and methods

While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can’t, therefore, say that one method is better than another

Professional software development

Professional software is intended for use by someone apart from its developer, is usually developed by teams rather than individuals. It is maintained and changed throughout its life. Software engineering is intended to support professional software development, rather than individual programming. It includes techniques that support program specification, design, and evolution.

Software engineers are concerned with developing software products (i.e., software which can be sold to a customer). There are two kinds of software products:

1. Generic products These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them
   Eg: software for PCs such as databases, word processors, drawing packages, and project-management tools.
2. Customized (or bespoke) products These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer.
   Eg: control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.
   In generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification.

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.

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
2. 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

The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product. There are four fundamental activities that are common to all software processes. These activities are:

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.

2. Software development, where the software is designed and programmed.

3. Software validation, where the software is checked to ensure that it is what the customer requires.

4. Software evolution, where the software is modified to reflect changing customer and market requirements.

Software engineering ethics

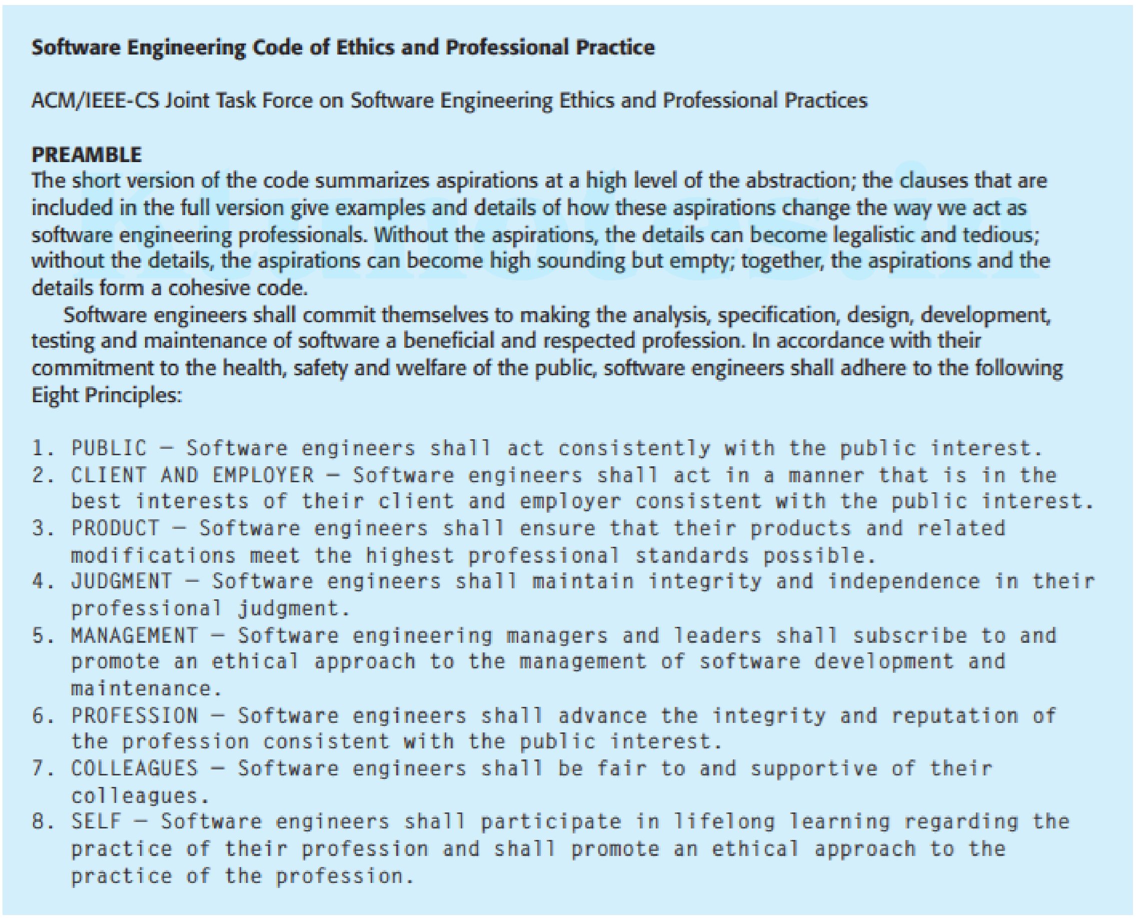
1. Confidentiality You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

2. Competence You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

3. Intellectual property rights You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.

4. Computer misuse You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses or other malware).

Professional societies and institutions have an important role to play in setting ethical standards. Organizations such as the ACM, the IEEE (Institute of Electrical and Electronic Engineers), and the British Computer Society publish a code of professional conduct or code of ethics. Members of these organizations undertake to follow that code when they sign up for membership.



Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE
The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

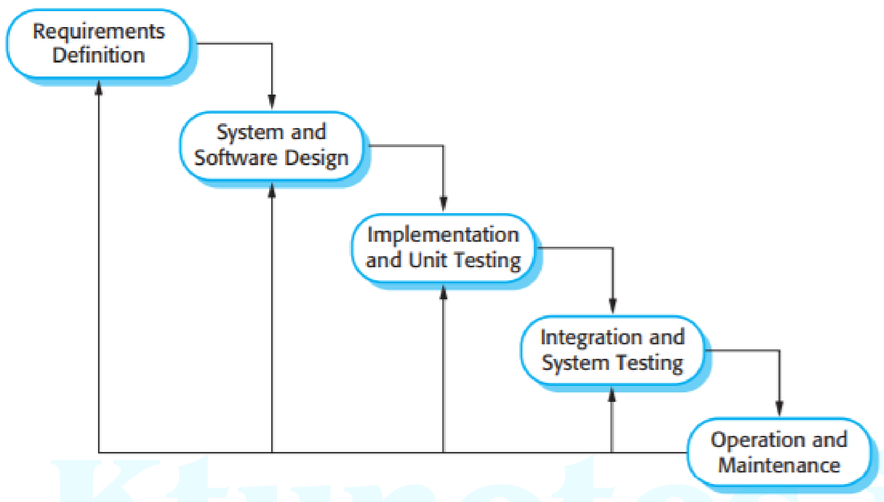
Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC — Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT — Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES — Software engineers shall be fair to and supportive of their colleagues.
8. SELF — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

A software process model is a simplified representation of a software process. Each process model represents a process from a particular perspective, and thus provides only partial information about that process.

The waterfall model

Because of the cascade from one phase to another, this model is known as the 'waterfall model' or software life cycle. 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 work on them.


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

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

1. 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.

2. System and software design :The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing :During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. 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.

5. Operation and maintenance :Normally (although not necessarily), 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.

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. During design, problems with requirements are identified. During coding, design problems are found and so on.

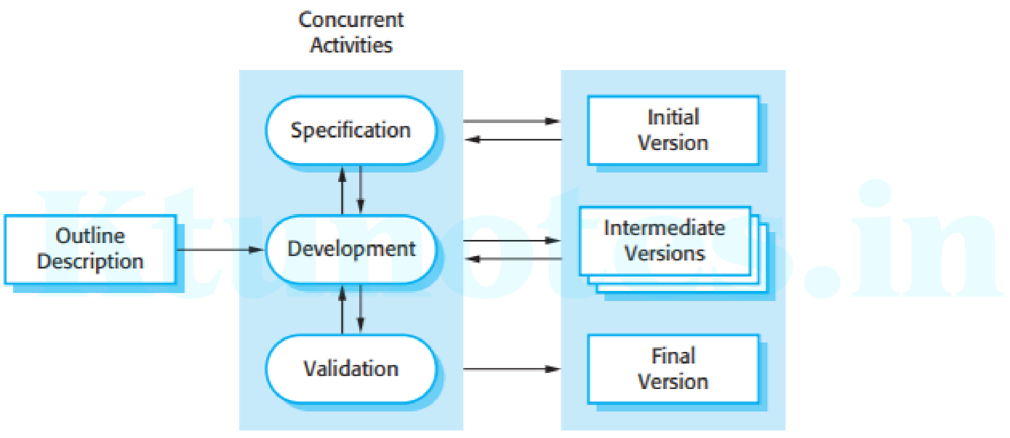
The software process is not a simple linear model but involves feedback from one phase to another. Documents produced in each phase may then have to be modified to reflect the changes made.

Because of the costs of producing and approving documents, iterations can be costly and involve significant rework.

the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.

Incremental development

Incremental development is based on 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.

Incremental software development is better than a waterfall approach for most business, e-commerce, and personal systems.

By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.

Each increment or version of the system incorporates some of the functionality that is needed by the customer. Customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

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

1. 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.

2. 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.

3. 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.

Incremental development is the most common approach for the development of application systems. This approach can be either plan-driven, agile, or, more usually, a mixture of these approaches. In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified but the development of later increments depends on progress and customer priorities.

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.

Process activities –

The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes.

1. 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. Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

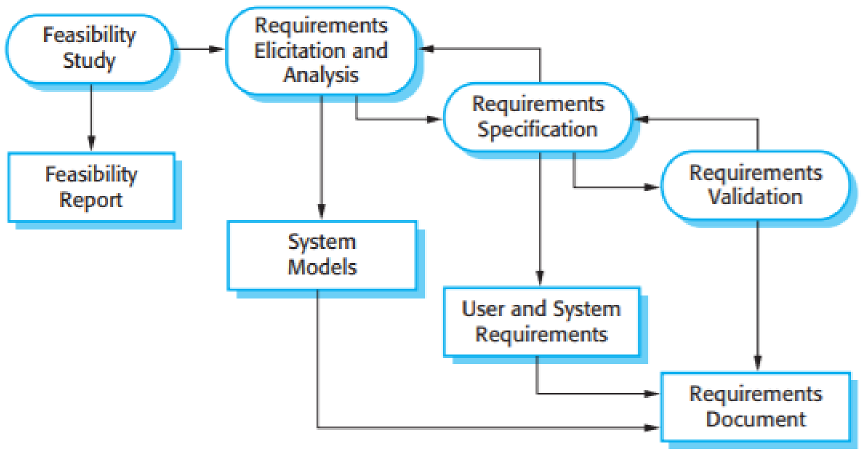


Figure: The requirements engineering process
The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.
Requirements are usually presented at two levels of detail:

- End-users and customers need a high-level statement of the requirements;
- System developers need a more detailed system specification.

There are four main activities in the requirements engineering process:

1. Feasibility study: An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

2. Requirements elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

3. Requirements specification: Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

4. Requirements validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

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 but, if an incremental approach to development is used, may also involve refinement of the software specification.

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. Designers do not arrive at a finished design immediately but develop the design iteratively.

Abstract model of the design process showing the inputs to the design process, process activities, and the documents produced as outputs from this process is shown below.

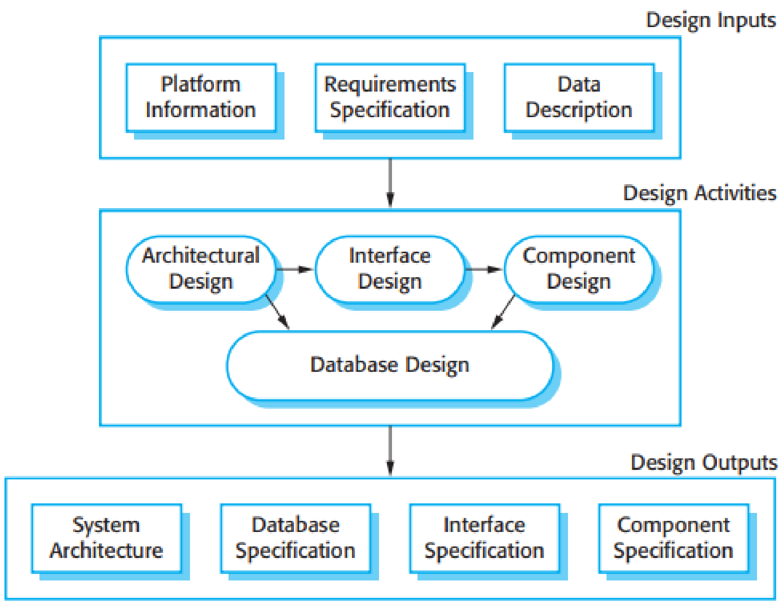


Fig: General model of the design process

Stages of the design process are sequential. In fact, design process activities are interleaved. Feedback from one stage to another and consequent design rework is inevitable in all design processes.

Most software interfaces with other software systems. These include the operating system, database, middleware, and other application systems. These make up the 'software platform', the environment in which the software will execute. Information about this platform is an essential input to the design process, as designers must decide how best to integrate it with the software's environment. The requirements specification is a description of the functionality the software must provide and its performance and dependability requirements. If the system is to process existing data, then the description of that data may be included in the platform specification; otherwise, the data description must be an input to the design process so that the system data organization to be defined

Four activities that may be part of the design process for information systems:

1. Architectural design: Identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships, and how they are distributed.

2. Interface design: Define the interfaces between system components. This interface specification must be unambiguous.

3. Component design: Take each system component and design how it will operate. The design model may be used to automatically generate an implementation.

4. Database design: Design the system data structures and how these are to be represented in a database. Again, the work here depends on whether an existing database is to be reused or a new database is to be created.

These activities lead to a set of design outputs. The detail and representation of these vary considerably. For critical systems, detailed design documents setting out precise and accurate descriptions of the system must be produced. If a model-driven approach is used, these outputs may mostly be diagrams. Where agile methods of development are used, the outputs of the design process may not be separate specification documents but may be represented in the code of the program.

The development of a program to implement the system follows naturally from the system design processes.

Programming is a personal activity and there is no general process that is usually followed. Some programmers start with components that they understand, develop these, and then move on to less-understood components. Others take the opposite approach, leaving familiar components till last because they know how to develop them

Programmers carry out some testing of the code they have developed. This often reveals program defects that must be removed from the program. This is called debugging. Defect testing and debugging are different processes. Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.

Software validation

Software validation or, more generally, verification and validation (V&V) 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.

Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development.

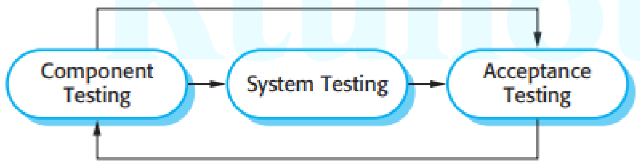
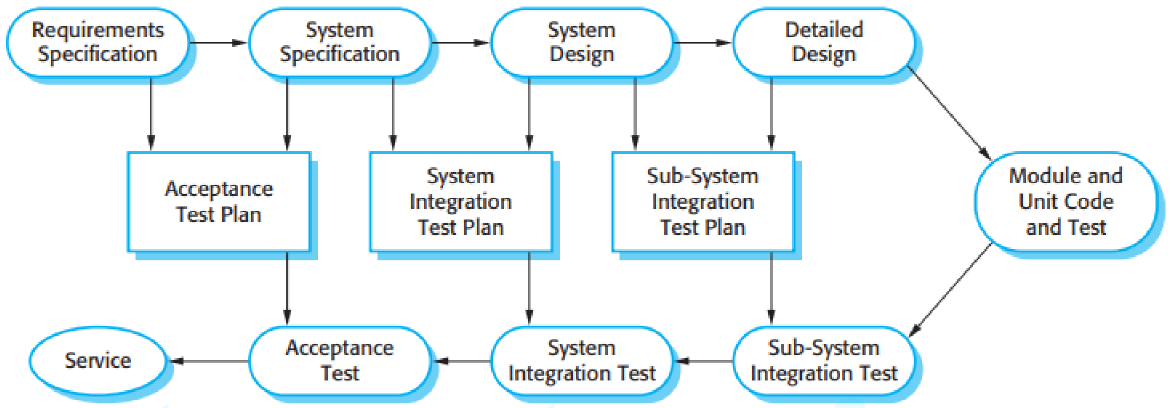


Fig: Stages of testing --three-stage testing process in which system components are tested then the integrated system is tested and, finally, the system is tested with the customer's data.

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. Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities. Test automation tools, such as JUnit, that can re-run component tests when new versions of the component are created, are commonly used.
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. It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties. For large systems, this may be a multi-stage process where components are integrated to form subsystems that are individually tested before these sub-systems are themselves integrated to form the final system.
3. Acceptance testing: This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition, because the real data exercise the system

in different ways from the test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.
Component development and testing processes are interleaved. Programmers make up their own test data and incrementally test the code as it is developed..
If an incremental approach to development is used, each increment should be tested as it is developed.
When a plan-driven software process is used (e.g., for critical systems development), testing is driven by a set of test plans. An independent team of testers works from these pre-formulated test plans, which have been developed from the system specification and design. Figure 2.7 illustrates how test plans are the link between testing and development activities. This is sometimes called the V-model of development.



Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

Acceptance testing is sometimes called 'alpha testing'. Custom systems are developed for a single client. The alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of the requirements.

When a system is to be marketed as a software product, a testing process called 'beta testing' is often used. Beta testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers. This exposes the product to real use and detects errors that may not have been anticipated by the system builders. After this feedback, the system is modified and released either for further beta testing or for general sale.
Software evolution

Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.

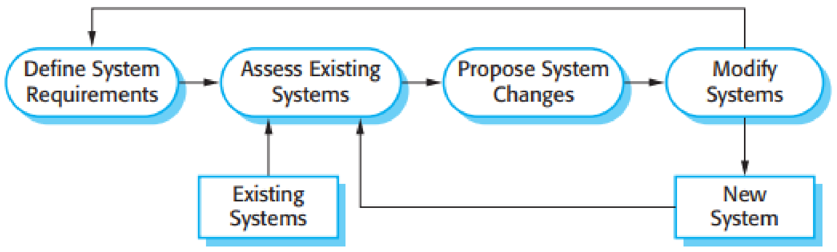


Figure: System evolution

Software engineering is an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

Coping with change

whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework.

There are two related approaches that may be used to reduce the costs of rework:

1. Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers. They can experiment with the prototype and refine their requirements before committing to high software production costs.
2. Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have to be altered to incorporate the change.

Two ways of coping with change and changing system requirements.

1. **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of some design decisions. This supports change avoidance as it allows users to experiment with the system before delivery and so refine their requirements. The number of requirements change proposals made after delivery is therefore likely to be reduced.
2. **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

Prototyping

A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions. Rapid, iterative development of the prototype is essential so that costs are controlled and system stakeholders can experiment with the prototype early in the software process.

A software prototype can be used in a software development process to help anticipate changes that may be required:

1. In the requirements engineering process, a prototype can help with the elicitation and validation of system requirements.

2. In the system design process, a prototype can be used to explore particular software solutions and to support user interface design.

System prototypes allow users to see how well the system supports their work. They may get new ideas for requirements, and find areas of strength and weakness in the software.

A system prototype may be used while the system is being designed to carry out design experiments to check the feasibility of a proposed design. For example, a database design may be prototyped and tested to check that it supports efficient data access for the most common user queries. Prototyping is also an essential part of the user interface design process.

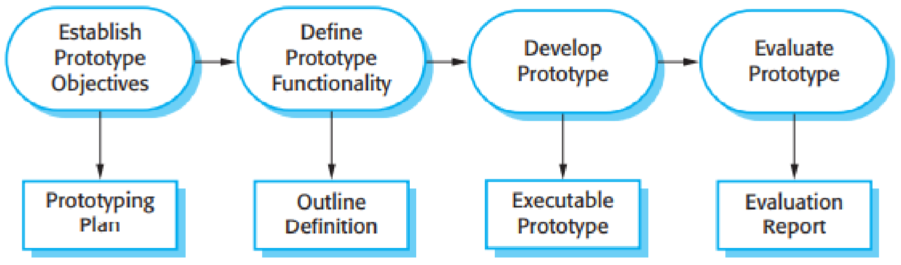


Figure: The process of prototype development

The process of prototype development

The objectives of prototyping should be made explicit from the start of the process. These may be to develop a system to prototype the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the feasibility of the application to managers. The same prototype cannot meet all objectives.

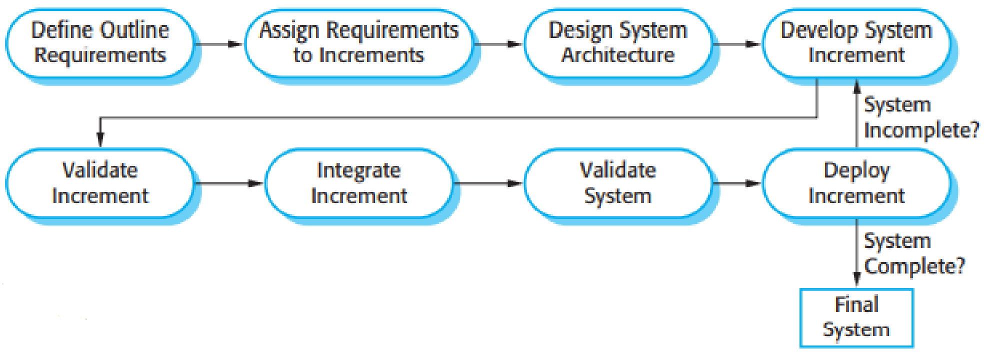
The next stage in the process is to decide what to put into and, perhaps more importantly, what to leave out of the prototype system. To reduce prototyping costs and accelerate the delivery schedule, you may leave some functionality out of the prototype.

The final stage of the process is prototype evaluation. Provision must be made during this stage for user training and the prototype objectives should be used to derive a plan for evaluation. Users need time to become comfortable with a new system and to settle into a normal pattern of usage. Once they are using the system normally, they then discover requirements errors and omissions.

Problems with prototyping

A general problem with prototyping is that the prototype may not necessarily be used in the same way as the final system. The tester of the prototype may not be typical of system users. The training time during prototype evaluation may be insufficient. If the prototype is slow, the evaluators may adjust their way of working and avoid those system features that have slow response times. When provided with better response in the final system, they may use it in a different way.

Incremental delivery



Incremental delivery is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment. In an incremental delivery process, customers identify, in outline, the services to be provided by the system. They identify which of the services are most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality. The allocation of services to increments depends on the service priority, with the highest-priority services implemented and delivered first. Once the system increments have been identified, the requirements for the services to be delivered in the first increment are defined in detail and that increment is developed. During development, further requirements analysis for later increments can take place but requirements changes for the current increment are not accepted.

Once an increment is completed and delivered, customers can put it into service. This means that they take early delivery of part of the system functionality. They can experiment with the system and this helps them clarify their requirements for later system increments. As new increments are completed, they are integrated with existing increments so that the system functionality improves with each delivered increment.

Incremental delivery has a number of advantages:

1. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments. Unlike prototypes, these are part of the real system so there is no re-learning when the complete system is available.

2. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.

3. The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.

4. As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing. This means that customers are less likely to encounter software failures in the most important parts of the system.

Problems with incremental delivery:

1. Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

2. Iterative development can also be difficult when a replacement system is being developed. Users want all of the functionality of the old system and are often unwilling to experiment with an incomplete new system. Therefore, getting useful customer feedback is difficult.

3. The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract. In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

Types of system where incremental development and delivery is not the best approach.

- Very large systems where development may involve teams working in different locations.

- Some embedded systems where the software depends on hardware development
- Some critical systems where all the requirements must be analyzed to check for interactions that may compromise the safety or security of the system.

These systems, of course, suffer from the same problems of uncertain and changing requirements. Therefore, to address these problems and get some of the benefits of incremental development, a process may be used in which a system prototype is developed iteratively and used as a platform for experiments with the system requirements and design. With the experience gained from the prototype, definitive requirements can then be agreed

MODULE 1

Introduction to Software Engineering - Professional software development, Software engineering ethics. Software process models - The waterfall model, Incremental development. Process activities - Software specification, Software design and implementation, Software validation, Software evolution. Coping with change - Prototyping, Incremental delivery, Boehm's Spiral Model. Agile software development - Agile methods, agile manifesto - values and principles. Agile development techniques, Agile Project Management. Case studies : An insulin pump control system. Mentcare - a patient information system for mental health care.

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation, and 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.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.

What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times, and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the Web made to software engineering?	The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Professional software development

- Professional software is intended for use by someone apart from its developer.
- Usually developed by teams rather than individuals.
- It is maintained and changed throughout its life.
- Software engineering is intended to support professional software development, rather than individual programming.
- It includes techniques that support program specification, design, and evolution.

- Software engineers are concerned with developing software products (i.e., software which can be sold to a customer). There are **two kinds of software products**:

Generic products

- These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them
- Eg: software for PCs such as databases, word processors, drawing packages, and project-management tools.

Customized (or bespoke) products

These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer.

- Eg: control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

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.

Software engineering is important for two reasons:

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- 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

Software Process

- The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product.
- There are **four fundamental activities that are common to all software processes**. These activities are:

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

Software engineering ethics

1. Confidentiality: Respect the confidentiality of employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

2. Competence: You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

3. Intellectual property rights: You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.

4. Computer misuse: You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses or other malware).

Software engineering ethics

- Professional societies and institutions have an important role to play in setting ethical standards.
- Organizations such as the ACM, the IEEE (Institute of Electrical and Electronic Engineers), and the British Computer Society publish a code of professional conduct or code of ethics.
- Members of these organizations undertake to follow that code when they sign up for membership.

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE
The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC — Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT — Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES — Software engineers shall be fair to and supportive of their colleagues.
8. SELF — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

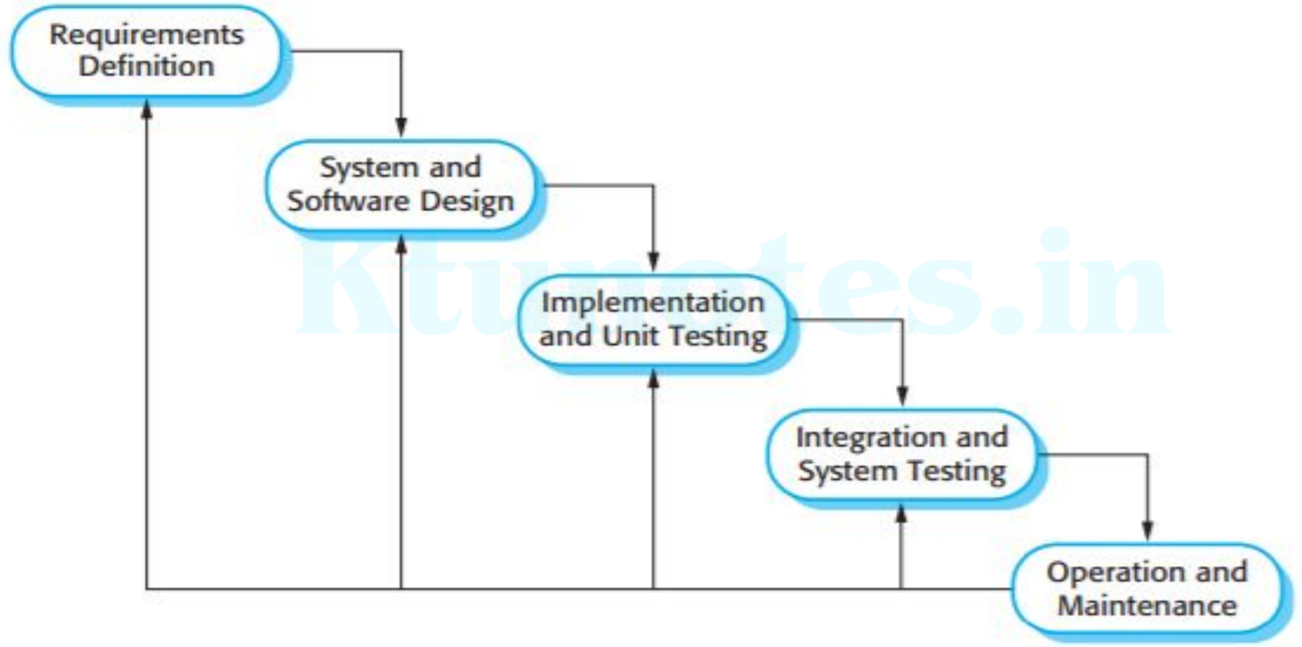
Software Process Models

- A software process model is a simplified representation of a software process. Each process model represents a process from a particular perspective, and thus provides only partial information about that process.

❑ Waterfall model

❑ Incremental model

The waterfall Model


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

The waterfall Model

- Because of the cascade from one phase to another, this model is known as the ‘waterfall model’ or software life cycle. 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 work on them.

The waterfall Model

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

1. 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.

2. System and software design :The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing :During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. 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.

5. Operation and maintenance : 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.

- The result of each phase is one or more documents that are approved (‘signed off’).

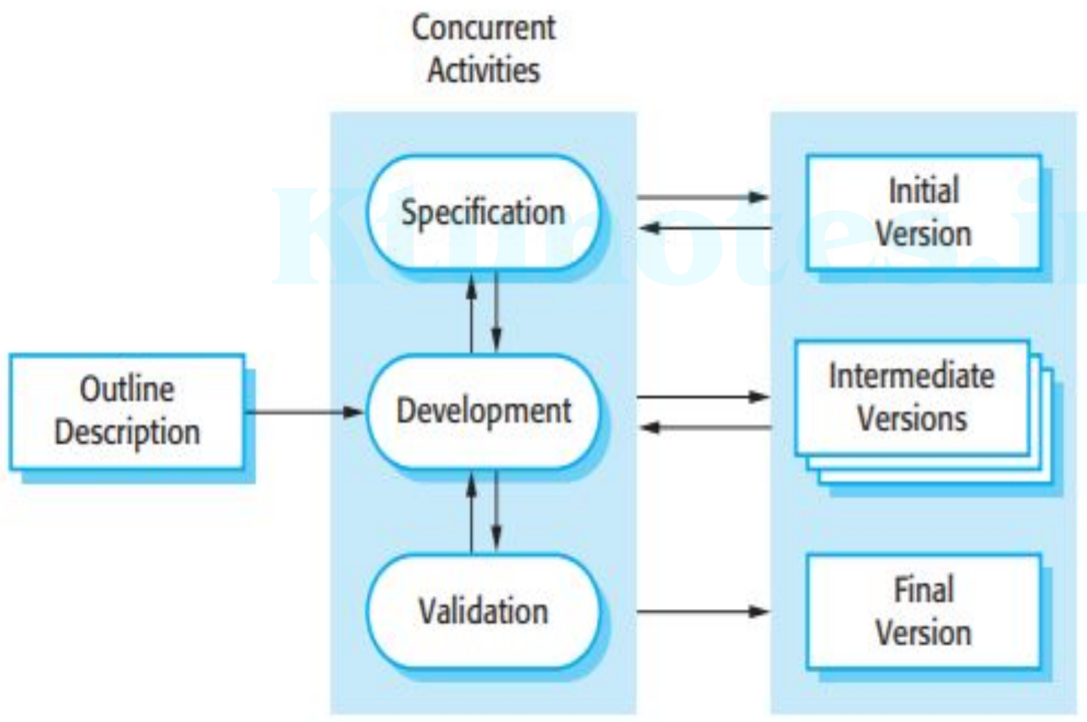
The waterfall Model

- The following phase should not start until the previous phase has finished.
- Because of the costs of producing and approving documents, iterations can be costly and involve significant rework.
- the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.

Incremental Development

- Incremental development is based on 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.

Incremental development



Incremental Development

- Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.
- Incremental software development is better than a waterfall approach for most business, e-commerce, and personal systems.
- By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.
- Each increment or version of the system incorporates some of the functionality that is needed by the customer.

Incremental development

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

1. 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.

2. It is easier to get customer feedback on the development work that has been done.

3. More rapid delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included.

Incremental Development

The incremental approach has two problems:

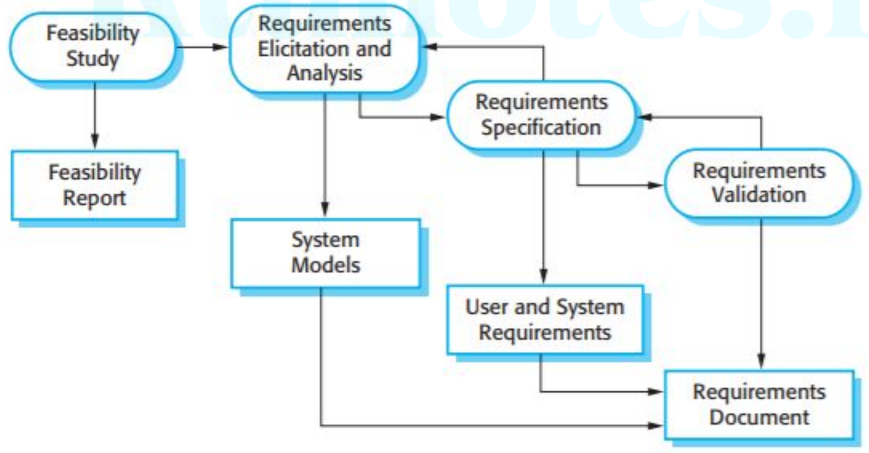
1. The process is not visible. Managers need regular deliverables to measure progress.
2. System structure tends to degrade as new increments are added. Incorporating further software changes becomes increasingly difficult and costly.

Process activities

- Software specification or requirements engineering
- Software Design and Implementation
- Software validation
- Software evolution

Software specification (Requirements engineering)

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.



Software specification

The requirements engineering process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.

Requirements are usually presented at two levels of detail:

- End-users and customers need a high-level statement of the requirements;
- System developers need a more detailed system specification.

Software specification

There are **four main activities in the requirements engineering process**:

1. Feasibility study: An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

2. Requirements elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes.

3. Requirements specification: Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document**. User requirements** are abstract statements of the system requirements for the customer and end-user of the system; **system requirements** are a more detailed description of the functionality to be provided.

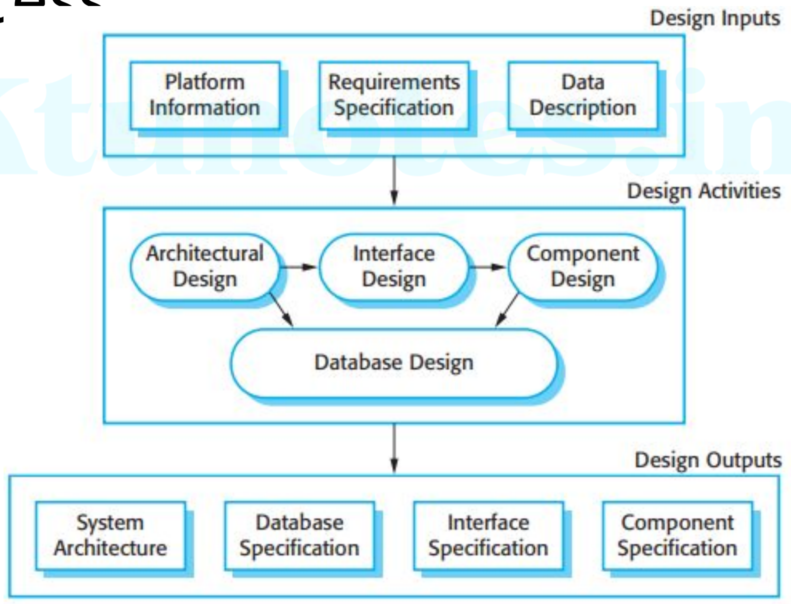
4. Requirements validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

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 but, if an incremental approach to development is used, may also involve refinement of the software specification
- 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.

Software Design and Implementation

- Fig: abstract model of the design process showing the inputs to the design process, process activities, and the documents produced as outputs from this process



Software Design and Implementation

- Stages of the design process are sequential. In fact, design process activities are interleaved. Feedback from one stage to another and consequent design rework is inevitable in all design processes.
- Four activities that may be part of the design process for information systems:

1. Architectural design: Identify the overall structure of the system, the principal components , their relationships, and how they are distributed.
2. Interface design: Define the interfaces between system components.
3. Component design: Take each system component and design how it will operate.
4. Database design: Design the system data structures and how these are to be represented in a database.

- These activities lead to a set of design outputs.

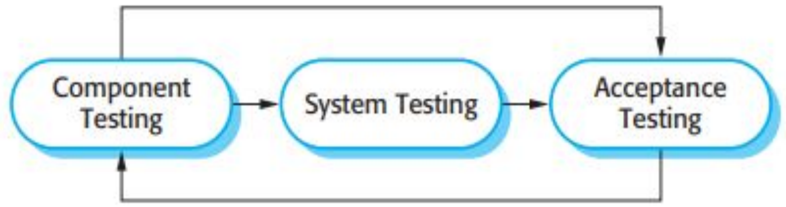
Software Design and Implementation

- The development of a program to implement the system follows naturally from the system design processes.
- Programming is a personal activity and there is no general process that is usually followed.
- Programmers carry out some testing of the code they have developed. This often reveals program defects that must be removed from the program. This is called debugging. Defect testing and debugging are different processes.

Software Validation

- Software validation or, more generally, verification and validation (V&V) 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.
- Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development.

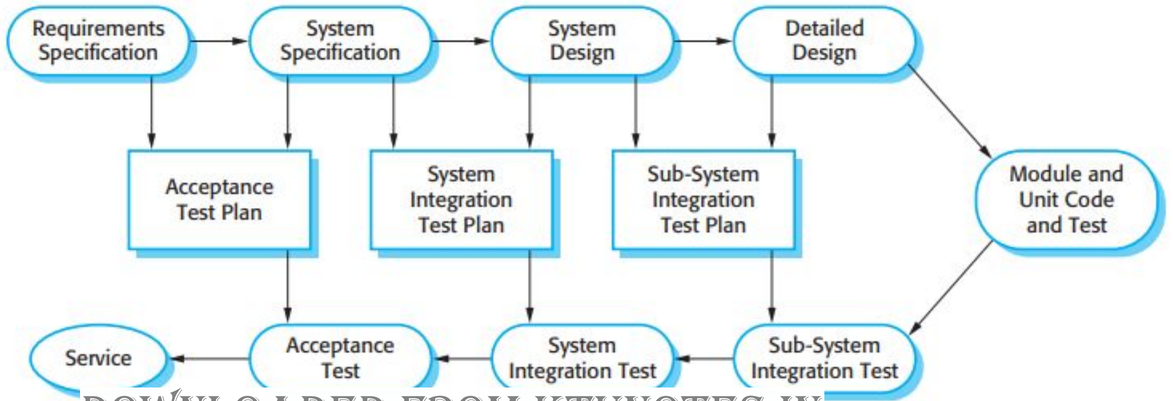
Software Validation



- Fig: Stages of testing --three-stage testing process in which system components are tested then the integrated system is tested and, finally, the system is tested with the customer’s data.
- 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.
- 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.
- Acceptance testing: This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data.

Software Validation

- If an **incremental approach** to development is used, each increment should be tested as it is developed.
- When a **plan-driven software process** is used (e.g., for critical systems development), testing is driven by a set of test plans. An independent team of testers works from these pre-formulated test plans, which have been developed from the system specification and design. This is sometimes called the **V-model of development**.

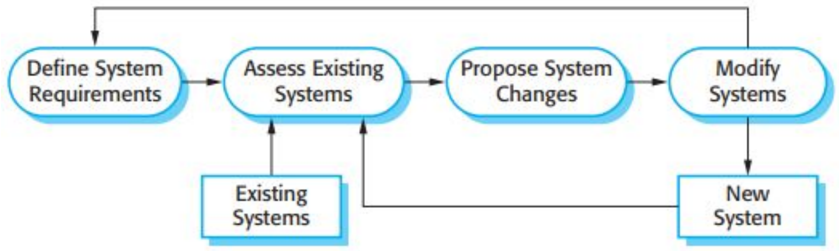

Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

Software Validation

- Acceptance testing is sometimes called ‘**alpha testing**’. Custom systems are developed for a single client. The alpha testing process continues until the system developer and the client agree that the delivered system is an acceptable implementation of the requirements.
- When a system is to be marketed as a software product, a testing process called ‘**beta testing**’ is often used. Beta testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers.

Software Evolution

- Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design.
- However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.
- software engineering is an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs



Coping with change

- Change is inevitable in all large software projects.
- whatever software process model is used, it is essential that it can accommodate changes to the software being developed.
- Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework.

Coping with change

There are two related **approaches that may be used to reduce the costs of rework:**

1. **Change avoidance**, where the software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers.

2. **Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed.

Coping with change

- **Two ways of coping with change and changing system requirements.**

1. **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of some design decisions. This ***supports change avoidance*** as it allows users to experiment with the system before delivery and so refine their requirements.
2. **Incremental delivery**, where system increments are delivered to the customer for comment and experimentation. This ***supports both change avoidance and change tolerance***.

The notion of refactoring, namely improving the structure and organization of a program, is also an important mechanism that supports change tolerance.

Prototyping

- A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions.

A software prototype can be used in a software development process to help anticipate changes that may be required:

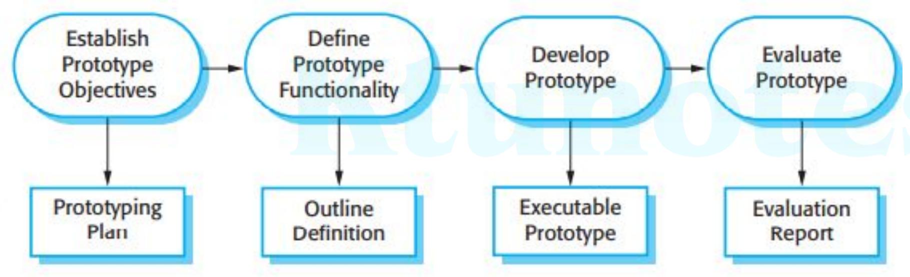
1. In the requirements engineering process, a prototype can help with the elicitation and validation of system requirements.
2. In the system design process, a prototype can be used to explore particular software solutions and to support user interface design.

Prototyping

- System prototypes allow users to see how well the system supports their work. They may get new ideas for requirements, and find areas of strength and weakness in the software.
- A system prototype may be used while the system is being designed to carry out design experiments to check the feasibility of a proposed design.
- Prototyping is also an essential part of the user interface design process.

Prototyping

- The process of prototype development

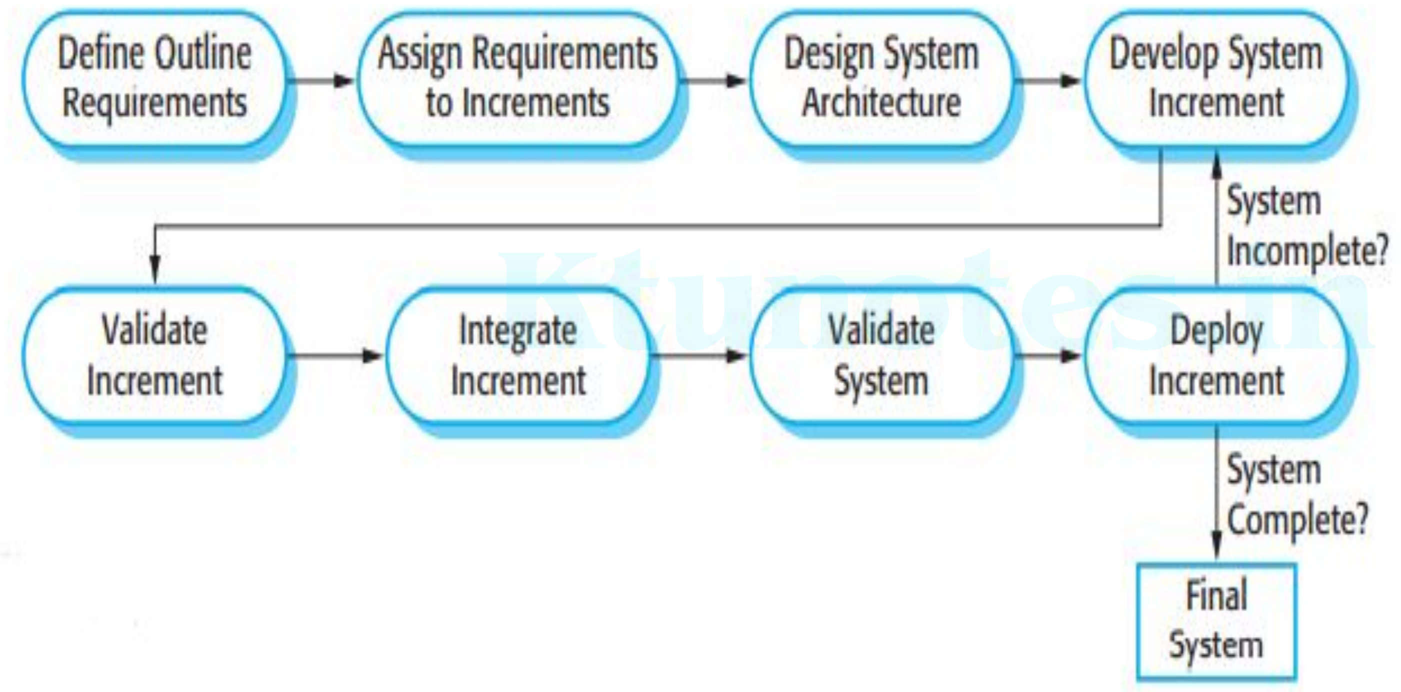


Prototyping

Problems with prototyping

- The prototype may not necessarily be used in the same way as the final system.
- The tester of the prototype may not be typical of system users.
- The training time during prototype evaluation may be insufficient.
- If the prototype is slow, the evaluators may adjust their way of working and avoid those system features that have slow response times.

Incremental Delivery



Incremental Delivery

- Incremental delivery is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in an operational environment.
- In an incremental delivery process, customers identify:
  - ❑ the services to be provided by the system
  - ❑ which of the services are most important and which are least important to them.
- A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality.
- The allocation of services to increments depends on the service priority, with the highest-priority services implemented and delivered first.
- Once the system increments have been identified, the requirements for the services to be delivered in the first increment are defined in detail and that increment is developed.
- During development, further requirements analysis for later increments can take place but requirements changes for the current increment are not accepted.

Incremental Delivery

- Once an increment is completed and delivered, customers can put it into service. This means that they take early delivery of part of the system functionality.
- Customers can experiment with the system and this helps them clarify their requirements for later system increments.
- As new increments are completed, they are integrated with existing increments so that the system functionality improves with each delivered increment.

Incremental Delivery

Advantages

1. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments. Unlike prototypes, these are part of the real system so there is no re-learning when the complete system is available.
2. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.
3. The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
4. As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing. This means that customers are less likely to encounter software failures in the most important parts of the system.

Incremental delivery

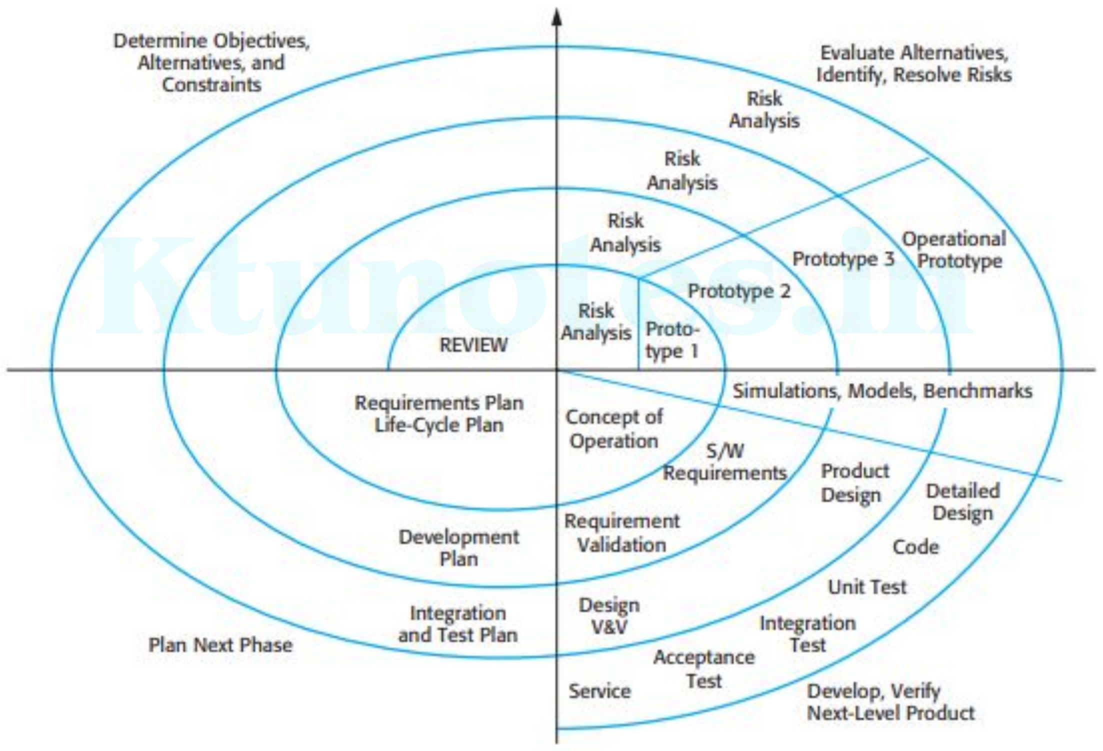
Types of system where incremental development and delivery is not the best approach.

- Very large systems where development may involve teams working in different locations.
- Some embedded systems where the software depends on hardware development
- Some critical systems where all the requirements must be analyzed to check for interactions that may compromise the safety or security of the system.

Boehm’s spiral model

- A risk-driven software process framework (the spiral model) was proposed by Boehm (1988).
- Here, the software process is represented as a spiral, rather than a sequence of activities with some backtracking from one activity to another.
- Each loop in the spiral represents a phase of the software process.
- The innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design, and so on.
- The spiral model combines change avoidance with change tolerance.
- It assumes that changes are a result of project risks and includes explicit risk management activities to reduce these risks.

Boehm’s spiral model



Boehm’s spiral model

Each loop in the spiral is split into four sectors:

1. Objective setting: Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.

2. Risk assessment and reduction: For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

3. Development and validation: After risk evaluation, a development model for the system is chosen. For example, throwaway prototyping may be the best development approach if user interface risks are dominant. If safety risks are the main consideration, development based on formal transformations may be the most appropriate process, and so on. If the main identified risk is sub-system integration, the waterfall model may be the best development model to use.

4. Planning: The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

Boehm’s spiral model

- The main difference between the spiral model and other software process models is its explicit recognition of risk.
- A cycle of the spiral begins by elaborating objectives such as performance and functionality. Alternative ways of achieving these objectives, and dealing with the constraints on each of them, are then enumerated.
- Each alternative is assessed against each objective and sources of project risk are identified.
- The next step is to resolve these risks by information-gathering activities such as more detailed analysis, prototyping, and simulation.
- Once risks have been assessed, some development is carried out, followed by a planning activity for the next phase of the process.
- Informally, risk simply means something that can go wrong.

Agile software development

Agile Methods:

- Allowed the development team to focus on the software itself rather than on its design and documentation.
- Agile methods universally rely on an incremental approach to software specification, development, and delivery.
- They are best suited to application development where the system requirements usually change rapidly during the development process.
- They are intended to deliver working software quickly to customers, who can then propose new and changed requirements to be included in later iterations of the system

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

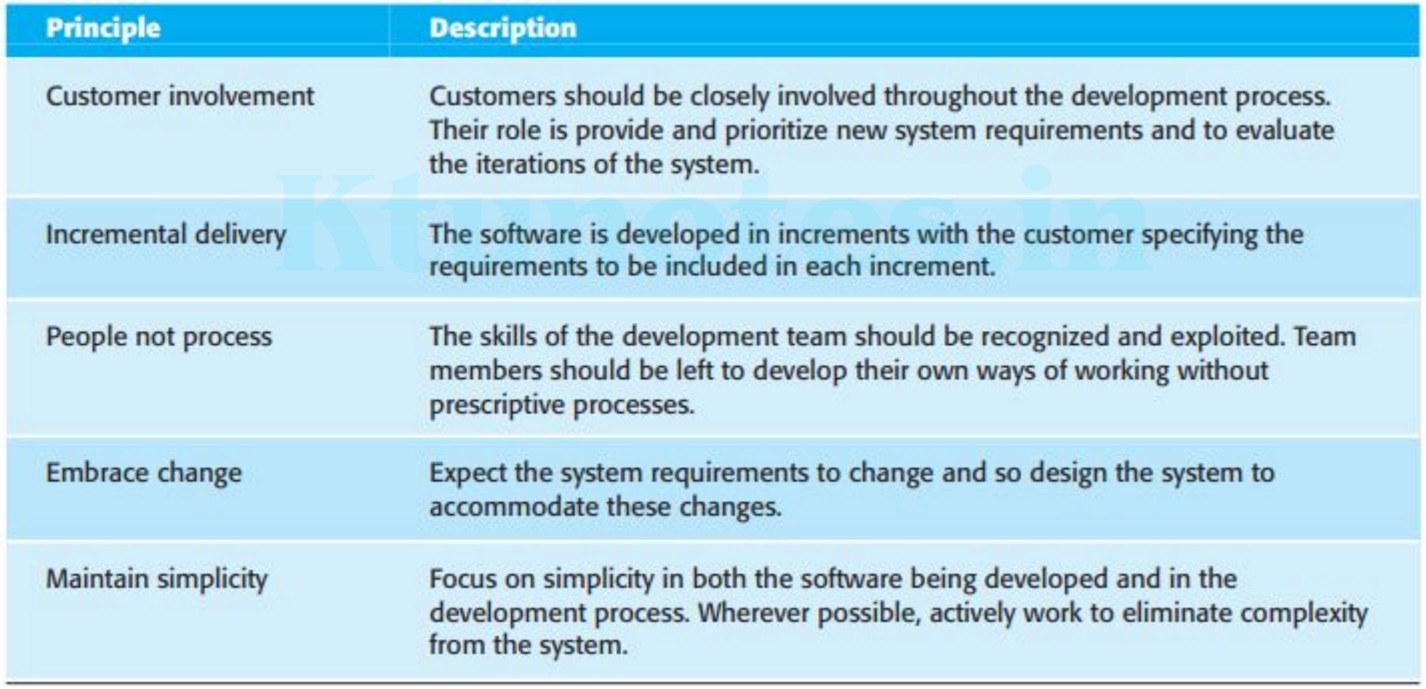
- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*
- *That is, while there is value in the items on the right, we value the items on the left more*

Best known agile methods:

- Extreme programming
- Scrum
- Crystal
- Adaptive software development
- Feature driven development

These agile methods are all based around the notion of incremental development and delivery, but propose different processes to achieve this. However, they share a set of principles.

The principle of agile methods



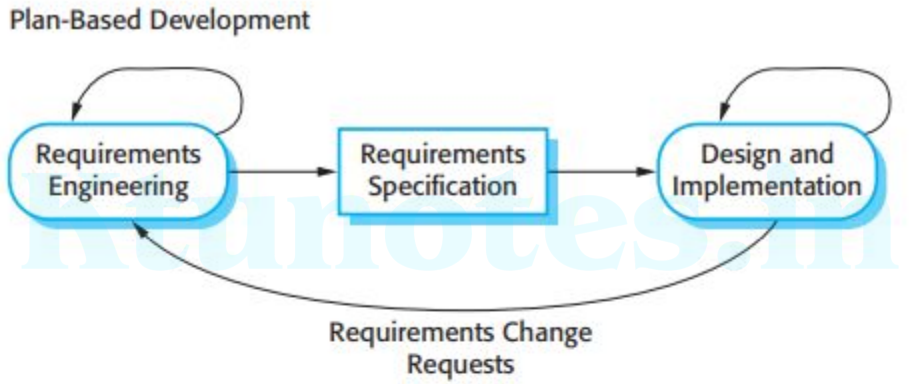
Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer 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.

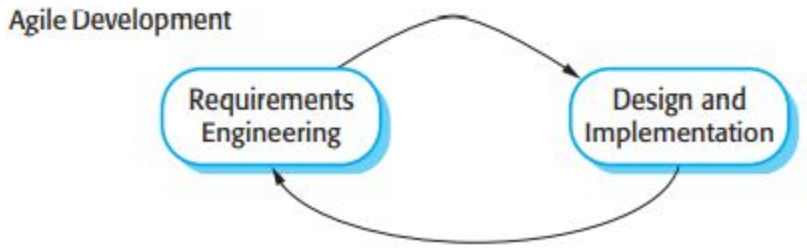
Agile methods have been very successful for some types of system development:

1. Product development where a software company is developing a small or medium-sized product for sale.
2. Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.

Plan-driven and agile development

- Agile approaches to software development consider design and implementation to be the central activities in the software process. They incorporate other activities, such as requirements elicitation and testing, into design and implementation.
- By contrast, a plan-driven approach to software engineering identifies separate stages in the software process with outputs associated with each stage


Plan-Based Development
Requirements Engineering
Requirements Specification
Design and Implementation
Requirements Change Requests


Agile Development
Requirements Engineering
Design and Implementation

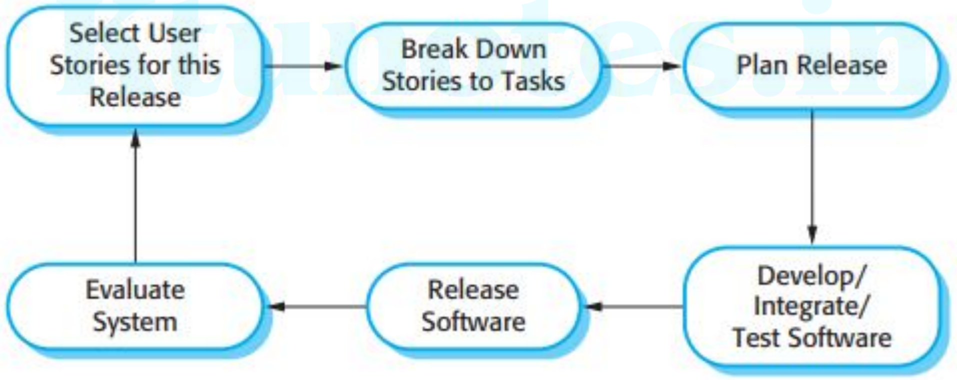
- There are various different processes and project management templates that you can apply to your projects to help you become agile. Extreme Programming (XP) and Scrum are the most popular.

Extreme programming

- Extreme programming (XP) is perhaps the best known and most widely used of the agile methods.
- The name was coined by Beck (2000) because the approach was developed by pushing recognized good practice, such as iterative development, to 'extreme' levels.

Extreme programming

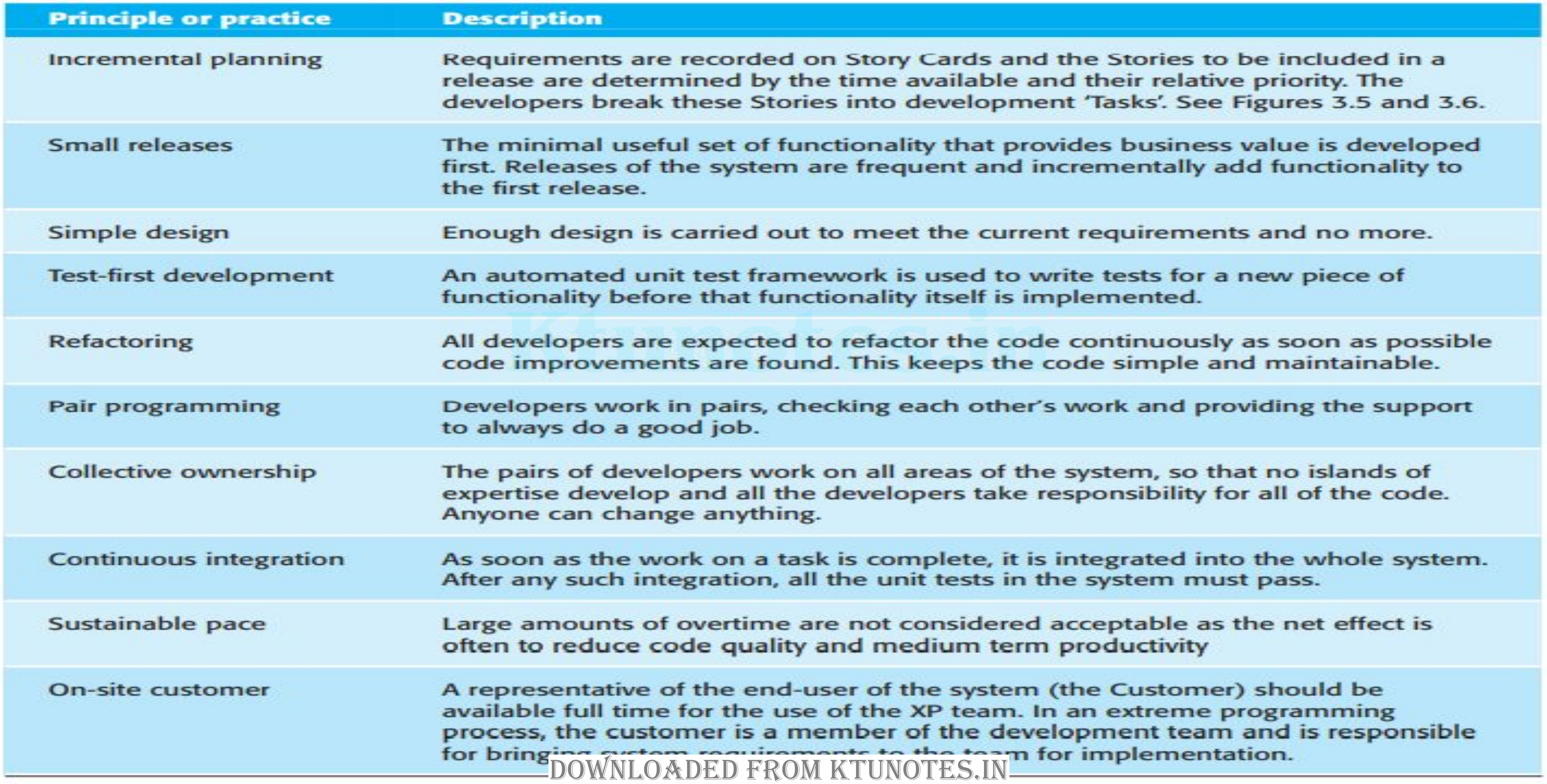
- For example, in XP, several new versions of a system may be developed by different programmers, integrated and tested in a day.


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Extreme programming

- In extreme programming, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks.
- Programmers work in pairs and develop tests for each task before writing the code.
- All tests must be successfully executed when new code is integrated into the system.
- There is a short time gap between releases of the system.

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'. See Figures 3.5 and 3.6.
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.
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.

Extreme programming

- The story cards are the main inputs to the XP planning process or the ‘planning game’.
- Once the story cards have been developed, the development team breaks these down into tasks and estimates the effort and resources required for implementing each task.
- This usually involves discussions with the customer to refine the requirements.
- The customer then prioritizes the stories for implementation, choosing those stories that can be used immediately to deliver useful business support.
- As requirements change, the unimplemented stories change or may be discarded. If changes are required for a system that has already been delivered, new story cards are developed and again, the customer decides whether these changes should have priority over new functionality.

Extreme programming

- Extreme programming takes an ‘extreme’ approach to incremental development.
- New versions of the software may be built several times per day and releases are delivered to customers roughly every two weeks.
- Release deadlines are never slipped; if there are development problems, the customer is consulted and functionality is removed from the planned release.
- When a programmer builds the system to create a new version, he or she must run all existing automated tests as well as the tests for the new functionality.
- The new build of the software is accepted only if all tests execute successfully.

Agile project management

- Agile project management
- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- agile development also has to be managed so that the best use is made of the time and resources available to the team.
- This requires a different approach to project management, which is adapted to incremental development and the particular strengths of agile methods.

Agile project management

- The **Scrum** approach is a general agile method but its focus is on managing iterative development rather than specific technical approaches to agile software engineering.
- There are three phases in Scrum.
  - The first is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - Finally, the project closure phase wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project.
  - The innovative feature of Scrum is its central phase, namely the sprint cycles. A Scrum sprint is a planning unit in which the work to be done is assessed, features are selected for development, and the software is implemented. At the end of a sprint, the completed functionality is delivered to stakeholders.

Agile project management

- Key characteristics of the scrum process are as follows:

1. Sprints are fixed length, normally 2–4 weeks. They correspond to the development of a release of the system in XP.

2. The starting point for planning is the product backlog, which is the list of work to be done on the project. During the assessment phase of the sprint, this is reviewed, and priorities and risks are assigned. The customer is closely involved in this process and can introduce new requirements or tasks at the beginning of each sprint.

3. The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.

4. Once these are agreed, the team organizes themselves to develop the software. Short daily meetings involving all team members are held to review progress and if necessary, reprioritize work. During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called **'Scrum master**'. The role of the Scrum master is to protect the development team from external distractions. The way in which the work is done depends on the problem and the team.

5. At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

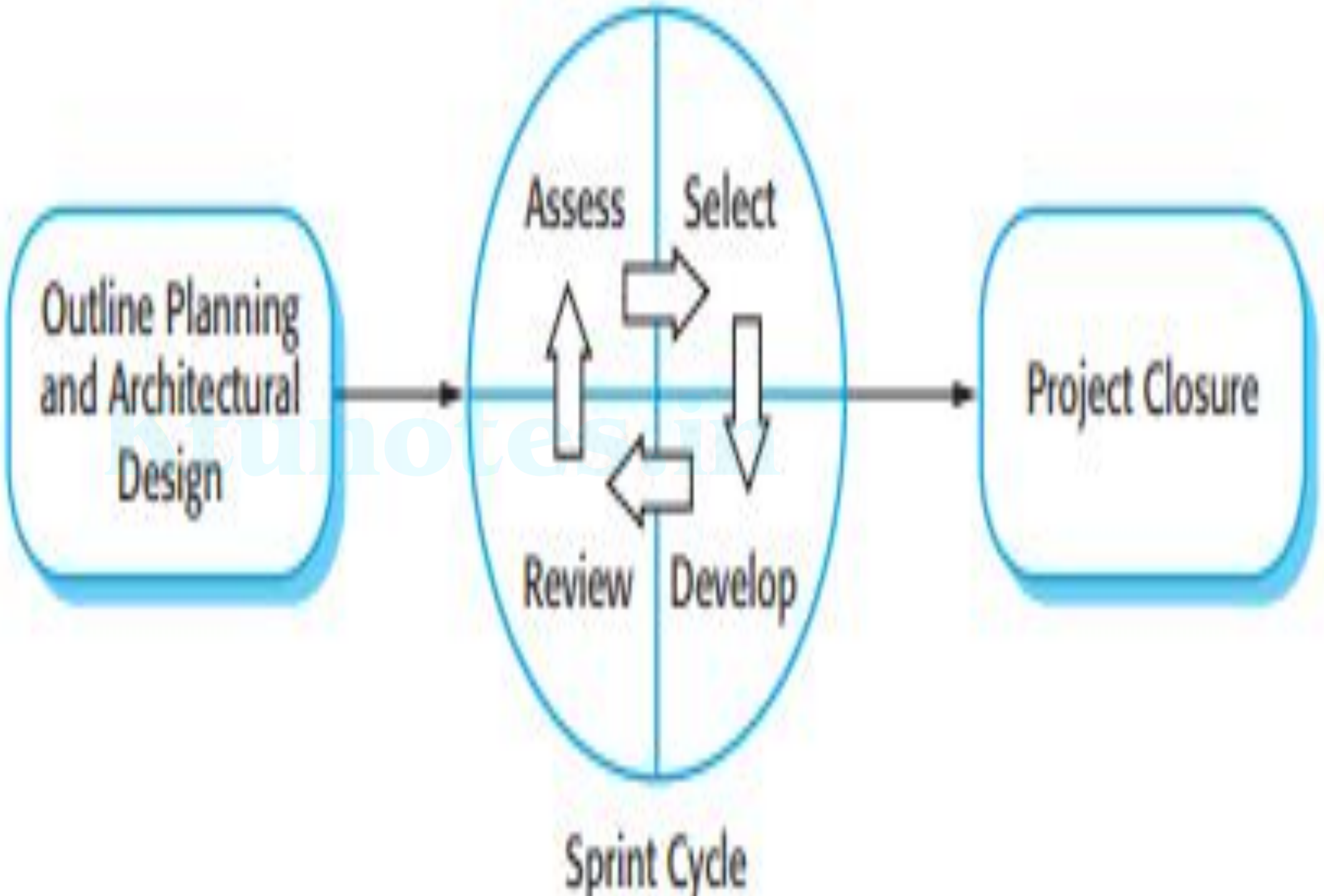
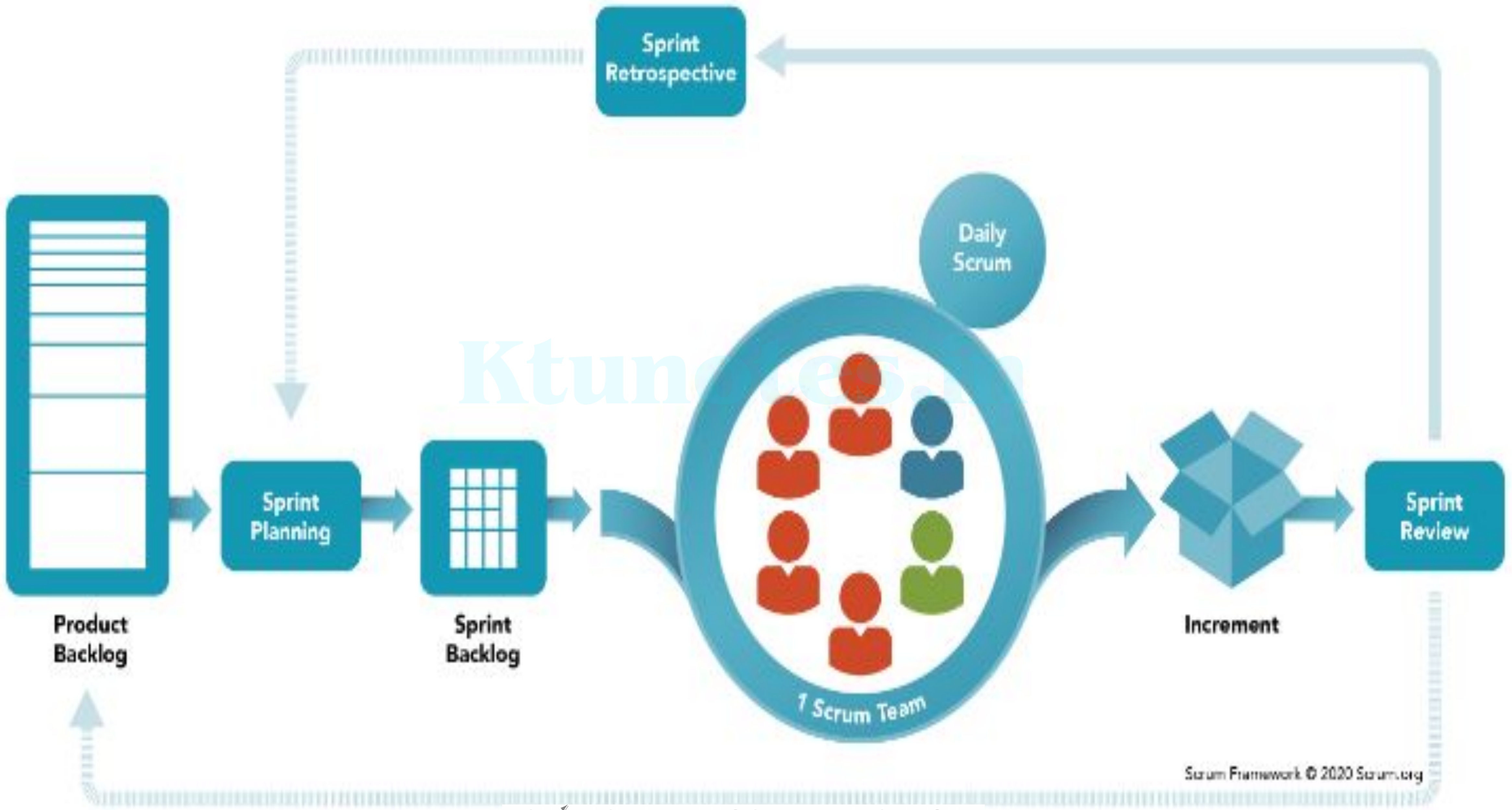


Figure 3.8 The Scrum process


Sprint Retrospective
Product Backlog
Sprint Planning
Sprint Backlog
Daily Scrum
1 Scrum Team
Increment
Sprint Review
Scrum Framework © 2020 Scrum.org

Advantages of scrum framework

1. The product is broken down into a set of manageable and understandable chunks.

2. Unstable requirements do not hold up progress.

3. The whole team has visibility of everything and consequently team communication is improved.

4. Customers see on-time delivery of increments and gain feedback on how the product works.

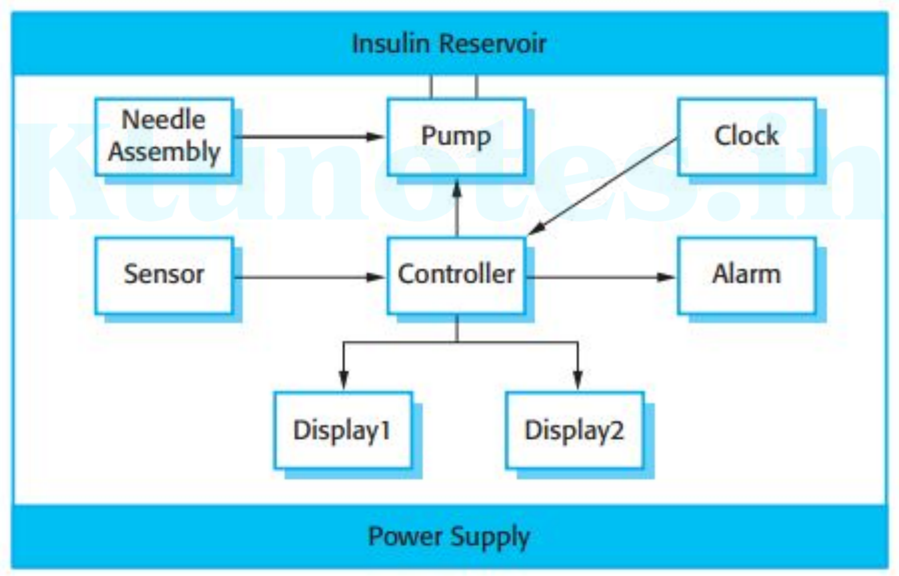
5. Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

CASE STUDIES

An insulin pump control system

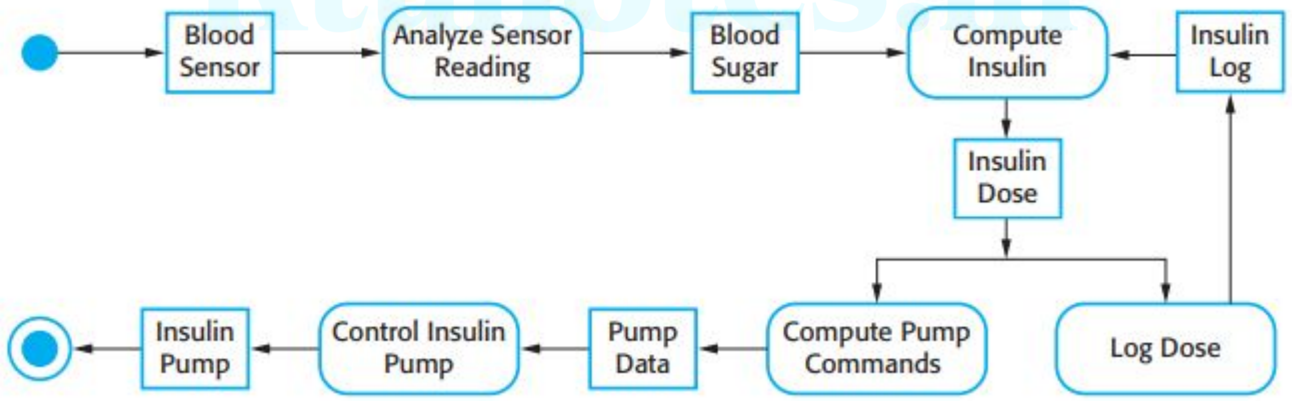
- An insulin pump is a medical system that simulates the operation of the pancreas.
- The software controlling this system is an embedded system, which collects information from a sensor and controls a pump that delivers a controlled dose of insulin to a user.
- A software-controlled insulin delivery system might work by using a microsensor embedded in the patient to measure some blood parameter that is proportional to the sugar level.
- This is then sent to the pump controller.
- This controller computes the sugar level and the amount of insulin that is needed.
- It then sends signals to a miniaturized pump to deliver the insulin via a permanently attached needle

Hardware components and organization of the insulin pump.



An insulin pump control system

- Figure: UML activity model that illustrates how the software transforms an input blood sugar level to a sequence of commands that drive the insulin pump.



An insulin pump control system

- This is a safety-critical system.
- If the pump fails to operate or does not operate correctly, then the user’s health may be damaged or they may fall into a coma because their blood sugar levels are too high or too low.
- There are, therefore, two essential high-level requirements that this system must meet:

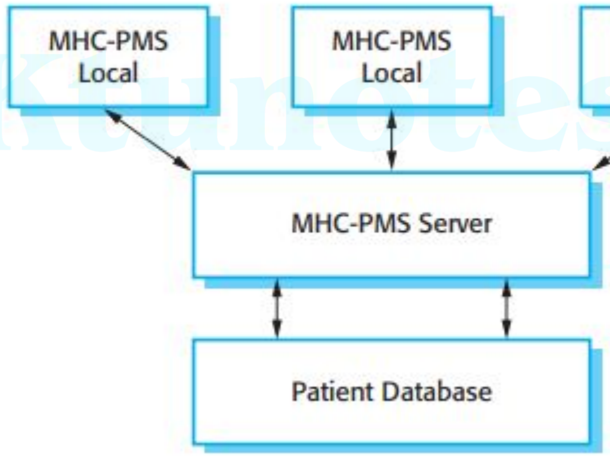
1. The system shall be available to deliver insulin when required.

2. The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.

A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.
- The system is not a complete medical records system so does not maintain information about other medical conditions. However, it may interact and exchange data with other clinical information systems.

The organization of the MHC-PMS



A patient information system for mental health care

The MHC-PMS has two overall goals:

1. To generate management information that allows health service managers to assess performance against local and government targets.

2. To provide medical staff with timely information to support the treatment of patients.

A patient information system for mental health care

- Users of the system include clinical staff such as doctors, nurses, and health visitors.
- Nonmedical users include receptionists who make appointments, medical records staff who maintain the records system, and administrative staff who generate reports.
- The system is used to record information about patients, consultations (date, doctor seen, subjective impressions of the patient, etc.), conditions, and treatments.
- Reports are generated at regular intervals for medical staff and health authority managers.
- staff always act in accordance with the laws(laws on data protection that govern the confidentiality of personal information and mental health laws) and that their decisions are recorded for judicial review if necessary.
- Privacy is a critical system requirement. It is essential that patient information is confidential and is never disclosed to anyone apart from authorized medical staff and the patient themselves.
- The MHC-PMS is a safety-critical system.
- The overall design of the system has to take into account privacy and safety requirements.