Overview of Software Engineering and the Software Development Process

CONTENTS

I. Definition of Software and Characteristics of Software

II. Types / Categories of Software
   1. System Software
   2. Real-time Software
   3. Business Software
   4. Engineering and Scientific Software
   5. Embedded Software
   6. Personal Computer Software
   7. Web-based Software

III. Software Engineering –
   1. Definition
   2. Need

IV. Relationship between System Engineering and Software Engineering

V. Software Engineering – A Layered Technology Approach

VI. Software Development Generic Process Framework
   1. Software Process
   2. Software Product
   3. Basic Framework Activities
   4. Umbrella Activities.

VII. Personal and Team Process Models (PSP and TSP)
   1. Concept
   2. Significance with respect to On-going Process Improvement
3. Goals
4. List of Framework Activities

VIII. Perspective Process Models
1. The Waterfall Model (Nature, Situations in which applicable with example, Associated Problems)
2. The Incremental Model (Nature, Situations in which applicable with example, General Steps, Drawbacks)
3. RAD Model (Nature, Situations in which applicable with example, General Steps, Drawbacks)
4. Prototyping (Nature, Situations in which applicable with example, General Steps, Drawbacks)
5. Spiral Model (Nature, Situations in which applicable with example, General Steps, Advantages, Disadvantages)

IX. Agile Software Development
6. Differences between Perspective and Agile Process Model
7. Features of the Agile Software Development Approach
8. Concept of Extreme Programming
I. Definition of Software and Characteristics of Software

1. Definition of Software

Software is defined as:
1. Instructions (computer programs) that when executed provide desired function and performance.
2. Data structures that enable the programs to adequately manipulate information.
3. Documents that describe the operation and use of the programs.

2. Characteristics of Software

(Question: Define software. Describe characteristics of software. – 3 Marks)

Software is a logical rather than a physical system element. Software has characteristics that are considerably different than those of hardware:

i. **Software is developed or engineered, it is not manufactured in the classical sense.**

   (Question: Justify your answer that, software is developed or engineered and not manufactured. – 3 Marks)

   1. Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
   2. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
   3. Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different.
   4. Both activities require the construction of a "product" but the approaches are different.
   5. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.
ii. Software doesn't "wear out."

(Question: Justify the implication that the software doesn't wear out but deteriorate. – 3 Marks)

1. Figure 1.1 explains the failure rate as a function of time for hardware.

![Bathtub Curve Diagram]

Figure 1.1

2. The relationship, called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life; defects are corrected and the failure rate drops to a steady-state level, ideally, quite low for some period of time.

3. With time the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.

4. Stated simply, the hardware begins to wear out.

5. Software is not susceptible to the environmental maladies that cause hardware to wear out.

6. The failure rate curve for software should take the form of the “idealized curve” shown in Figure 1.2.

7. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.

8. The idealized curve is a gross oversimplification of actual failure models for software.

9. The implication is clear—software doesn't wear out. But it does deteriorate!
iii. Although the industry is moving toward component-based assembly, most software continues to be custom built.

1. Consider the manner in which the control hardware for a computer-based product is designed and built. The design engineer draws a simple schematic of the digital circuitry, does some fundamental analysis to assure that proper function will be achieved, and then goes to the shelf where catalogs of digital components exist. Each integrated circuit (called an IC or a chip) has a part number, a defined and validated function, a well-defined interface, and a standard set of integration guidelines. After each component is selected, it can be ordered off the shelf.

2. A software component should be designed and implemented so that it can be reused in many different programs. In the 1960s, we built scientific subroutine libraries that were reusable in a broad array of engineering and scientific applications. These subroutine libraries reused well-defined algorithms in an effective manner but had a limited domain of application. Today, we have extended our view of reuse to encompass not only algorithms but also data structure. Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today’s graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.
II. Types / Categories of Software

1. System Software
   1. System software is a collection of programs written to service other programs.
   2. Few examples of system software are compilers, editors, and file management utilities, process complex, but determinate, information structures.
   3. Other systems applications are operating system components, drivers, and telecommunications.

2. Real-time Software
   (Question: Explain the features of real world software. – 3 Marks)
   1. Software that monitors or analyzes or controls real-world events as they occur is called real time.
   2. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application.
   3. A control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response can be maintained.

3. Business Software
   1. Business information processing is the largest single software application area. Discrete "systems”.
   2. For example: payroll, accounts receivable/payable, inventory have evolved into management information system (MIS) software that accesses one or more large databases containing business information.
   3. Applications in this area restructure existing data in a way that facilitates business operations or management decision making.
   4. In addition to conventional data processing application, business software applications also encompass interactive computing.

4. Engineering and Scientific Software
   1. Engineering and scientific software have been characterized by "number crunching" algorithms.
   2. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
   3. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms.
4. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

5. Embedded Software
   1. Intelligent products have become commonplace in nearly every consumer and industrial market.
   2. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
   3. Embedded software can perform very limited and esoteric functions, for example: keypad control for a microwave oven.
   4. To provide significant function and control capability, for example: digital functions in an automobile such as fuel control, dashboard displays, and braking systems.

6. Personal Computer Software
   1. The personal computer software market has burgeoned over the past two decades.
   2. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business applications, external network, and database access are only a few of hundreds of applications.

7. Web-based Software
   1. The Web pages retrieved by a browser are software that incorporates executable instructions and data.

III. Software Engineering

1. Definition:

   Software Engineering is the study and application of engineering to the design, development, and maintenance of software. Software engineering can be divided into ten sub disciplines. They are:
   1. **Software requirements**: The elicitation, analysis, specification, and validation of requirements for software.
   2. **Software design**: The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.
   3. **Software construction**: The detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging.
   4. **Software testing**: The dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.
5. **Software maintenance**: The totality of activities required to provide cost-effective support to software.

6. **Software configuration management**: The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle.

7. **Software engineering management**: The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.

8. **Software engineering process**: The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.

9. **Software engineering tools and methods**: The computer-based tools that are intended to assist the software life cycle processes, see Computer Aided Software Engineering, and the methods which impose structure on the software engineering activity with the goal of making the activity systematic and ultimately more likely to be successful.

10. **Software quality**: The degree to which a set of inherent characteristics fulfills requirements.

**2. Need of Software Engineering**

1. Computer information and control systems are increasingly embedded and integrated into the fabric of human society.

2. Computer systems are not standalone, but are usually just component parts of much larger, complex systems involving hardware, software, people, and all the unpredictable events in the natural world.

3. Our very lives depend on these interdependent systems working reliably all the time.

4. Most people, even most computer science graduates, are not fully aware of both the difficulty involved in building such complex systems and the essential need for those building them to be equipped with advanced techniques not taught in ordinary computer programming courses.

5. Software Engineering is the discipline dedicated to the principles and techniques required for the sound construction of the computer systems of today and tomorrow.

A software engineer must be equipped with techniques to

i. Model and understand complex interactive systems.
ii. Identify how computer information systems can be made to improve such systems.

iii. Manage the construction of the information system components.

iv. Ensure that procedures are in place for the continual testing and maintenance of operational systems.

IV. Relationship between System Engineering and Software Engineering

(Question: What is Software engineering? How it differs from system engineering? – 3 Marks)

1. A software engineer works on the software, while a system engineer will just work on system.

2. They are different because the system could be the BIOS, operating system, hard drive, memory, and other things. While a software engineer will only work on the programs that the user uses.

3. Software engineering is a branch to make software this is a continuous task, while software system is a product which is made by the software engineering.

![Figure 1.3](image.png)
V. **Software Engineering – A Layered Approach**  

*Question: Explain software engineering as a layered approach? – 6Marks*

Software engineering deals with process, methods their implementation tools and finally the quality of the product. This is known as the layered approach of software engineering as shown in Figure 1.4:

![Diagram of Software Engineering Layers](image)

**Figure 1.4**

1. **Process, Methods and Tools**
   1. Software engineering is a layered technology. Referring to Figure 1.4, any engineering approach must rest on an organizational commitment to quality.
   2. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering.
   3. The bedrock that supports software engineering is a quality focus.

2. **Process Layer**
   1. The foundation for software engineering is the process layer.
   2. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
   3. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology.

3. **Method Layer**
   1. Software engineering methods provide the technical how-to's for building software.
   2. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.
4. Tools Layer
1. Software engineering tools provide automated or semi-automated for the process and the methods.
2. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database.

VI. Software Development Generic Process Framework
1. **Software Process:** A software process can be characterized as shown in Figure 1.5. A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
2. **Software Product:** A number of task sets—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
3. **Basic Framework Activities:** Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement to overlay the process model.
4. **Umbrella Activities:** Umbrella activities are independent of any one framework activity and occur throughout the process.

![Common process framework](image-url)
1. **Level 1:** Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

2. **Level 2:** Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. **Level 3:** Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization’s process for developing and supporting software. This level includes all characteristics defined for level 2.

4. **Level 4:** Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

5. **Level 5:** Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

VII. **Personal and Team Process Models (PSP and TSP)**

*(Question: Compare PSP and TSP with respect to software engineering - 4 Marks)*

1. **What is PSP?**
   1. PSP (Personal Software Process) provide a standard personal process structure for software developers. PSP consists of a set of methods, tables, scripts etc. that serve as a guideline for software developers to plan, measure and manage their work, including how to define their processes and measure quality and productivity.
   2. PSP provides software engineers a disciplined methods for improving personal software development process.
   3. PSP helps software engineers to
      i. Improve their estimating and planning skill.
      ii. Make commitment they can keep.
      iii. Manage the quality of their product.
      iv. Reduce the number of defects in their work.
   4. PSP represents metrics based approach to software engineering.
   5. PSP deals with software engineers to identify the errors early and to understand the types of errors.
2. **PSP Activities**  
*(Question: Describe various activities of PSP, - 6 Marks)*

1. **Planning:** This activity isolates requirements and specifications to be decided prior to the development and estimates the size and the cost of the project.

2. **High Level Design Review:** Formal verification methods are applied to uncover errors.

3. **Postmortem:** Metrics and measures should be providing guidance for modification.

4. **Development:** The component level design is reviewed and refined. The code is generated, reviewed and tested.

5. **High-Level Design:** External specifications and requirements for each component to be constructed and developed.

3. **Goals of PSP**

   The goal of PSP is to provide software engineers with disciplined methods for improving personal software development.

   **PSP helps software engineers to**
   
   i. Improve their estimating and planning skill.
   
   ii. Make commitment they can keep.
   
   iii. Manage the quality of their product.
   
   iv. Reduce the number of defects in their work.

4. **Disadvantage of PSP**

   1. PSP is intellectually challenging and demands a level of commitment which is always not possible to obtain.
   
   2. Training for PSP is lengthy and costs for the training is high.
   
   3. Required level of measurement is culturally difficult.

5. **What is TSP?**

   1. TSP (Team Software Process) is a guideline for software product development teams.
   
   2. TSP focuses on helping development teams to improve their quality and productivity to better meet goals of cost and progress.
   
   3. TSP is designed for groups ranging from 2 persons to 20 persons. TSP can be applied to large multiple-group processes for up to 150 persons.
   
   4. There are 8 steps for implementing PSP and TSP. Each step is focused on solving particular process problems.
6. **Goal of TSP?**
   1. To provide improvement guidance to high maturity organization.
   2. To facilitate university teaching of industrial grade team skills.
   3. To show managers how to motivate and coach their teams and how to sustain peak performance.

7. **Activities of TSP?**
   *(Question: Describe various activities of TSP, - 6 Marks)*
   1. **Launch:** It reviews course objective and describes the TSP structure and content. It assigns need and roles to the students and describes the customers need statement.
   2. **Strategy:** It creates a conceptual design for the product and establishes the development strategy.
   3. **Plan:** It estimates the size of each module to be developed. Planning also identifies tasks to be performed, and estimates the time to complete each task.
   4. **Requirement:** Analyzes the need statement and interviews the customer, specify and inspect the requirements.
   5. **Design:** it creates the high level design, specify the design, inspect the design and develop the integration plan.
   6. **Implement:** This uses the PSP to implement the modules and the functions.
   7. **Test:** Testing builds and integrates the system.
   8. **Postmortem:** Writes the cycle report and produces peer and team review.

VIII. **Perspective Process Models**
   a. **Waterfall Model**
      *(Question: Describe various phases of waterfall process model with neat diagram, - 6 Marks)*
      1. Also known as the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.
      2. Figure 1.6 illustrates the linear sequential model for software engineering.
      3. Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities:
4. System/information engineering and modeling. Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software.

5. Phases of waterfall model:
   i. **Software requirements analysis**: The requirements gathering process is intensified and focused specifically on software.
   
   ii. **Design Software**: Design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.
   
   iii. **Code generation**: The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.
   
   iv. **Testing**: Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

6. Advantages
   i. Organized approach, provides robust separation of phases.
   
   ii. Reflects common engineering practice.

7. Disadvantages
   i. Doesn’t cope well with changes the client.
   
   ii. Development required by teams might wait for each other.
   
   iii. A working version of the product is available only late.
8. Applicability
   i. When requirements are well known and few changes are likely to be needed.
   ii. Can be used also for parts of larger software systems.

b. Incremental Model
   *(Question: Describe various phases of Incremental process model with neat diagram, - 6 Marks)*
   1. The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping.
   2. Figure 1.7, the incremental model applies linear sequences in a staggered fashion as calendar time progresses.
   3. Each linear sequence produces a deliverable “increment” of the software.
   4. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.
   5. It should be noted that the process flow for any increment can incorporate the prototyping paradigm.

   ![Incremental Model Diagram](image)

   **Figure 1.7**

   6. The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature.
   7. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.
8. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.

9. Advantages:
   i. Provides better support for process iteration.
   ii. Reduces rework in the software construction process.
   iii. Some decisions on requirements may be delayed.
   iv. Allows early delivery of parts of the system.
   v. Supports easier integration of sub-systems.
   vi. Lower risk of project failure.
   vii. Delivery priorities can be more easily set.

10. Disadvantages:
   i. Increments need be relatively small
   ii. Mapping requirements to increments may not be easy.
   iii. Common software facilities may be difficult to identify.

11. Applicability:
   i. When it is possible to deliver the system “part-by-part”.

c. RAD Model

(Question: Describe various phases of RAD process model with neat diagram, - 6 Marks)

1. Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.

2. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.

3. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days).

4. Data modeling. The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called attributes) of each object are identified and the relationships between these objects defined.

5. Process modeling. The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
6. Application generation. RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary).

7. In all cases, automated tools are used to facilitate construction of the software.

8. Testing and turnover. Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

![Diagram](image)

**Figure 1.8**

---

d. **Prototyping**

*Question: Describe various phases of prototyping process model with neat diagram, - 6 Marks*

1. The prototyping paradigm (Figure 1.9) begins with requirements gathering.
2. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.

3. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats).
4. The quick design leads to the construction of a prototype.
5. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

Prototyping can also be problematic for the following reasons:
1. The customer sees what appears to be a working version of the software, unaware that the prototype is held together “with chewing gum and baling wire,” unaware that in the rush to get it working no one has considered overall software quality or long-term maintainability.
2. When informed that the product must be rebuilt so that high levels of quality can be maintained, the customer cries foul and demands that "a few fixes" be applied to make the prototype a working product.
3. Too often, software development management relents.
4. The developer often makes implementation compromises in order to get a prototype working quickly.
5. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.
6. After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate.
7. The less-than-ideal choice has now become an integral part of the system.
e. **Spiral Model**

*(Question: Describe various phases of Spiral process model with neat diagram, - 6 Marks)*

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Figure 1.10 depicts a spiral model that contains six task regions:

1. **Customer communication**—tasks required to establish effective communication between developer and customer.
2. **Planning**—tasks required to define resources, timelines, and other project-related information.
3. **Risk analysis**—tasks required to assess both technical and management risks.
4. **Engineering**—tasks required to build one or more representations of the application.
5. **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

![Figure 1.10](image)

6. **Advantages:**
   i. Risk reduction mechanisms are in place
   ii. Supports iteration and reflects real-world practices
   iii. Systematic approach
7. **Disadvantages:**
   i. Requires expertise in risk evaluation and reduction
   ii. Complex, relatively difficult to follow strictly
   iii. Applicable only to large systems
8. **Applicability:**
i. Internal development of large systems

IX. Agile Software Development

f. Difference between Perspective and Agile Process Model

(Question: Difference between Prescriptive and Agile Process Model - 3 marks)

1. The Agile movement proposes alternatives to traditional project management.
2. Agile approaches are typically used in software development to help businesses respond to unpredictability.
3. It’s easy to the problems with “waterfall” methodology.
4. It assumes that every requirement of the project can be identified before any design or coding occurs.
5. At the end of a project, a team might have built the software it was asked to build, but, in the time it took to create, business realities have changed so dramatically that the product is irrelevant.
6. Today very view organizations openly admit to doing waterfall or traditional command and control. But those habits persist.

Problems with Agile Method:

1. It can be difficult to keep the interest of customers who are involved in the process.
2. Team members may be unsuited to the intense involvement that characterizes agile method.
3. Prioritizing changes can be difficult where there are multiple stakeholders.
4. Maintaining simplicity requires extra work.
5. Contracts may be a problem as with other approaches to iterative development.

g. Features of the Agile Software Development Approach

(Question: Give the Features of the Agile Software Development Approach - 6 marks)

<table>
<thead>
<tr>
<th>1. Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Customer</td>
<td>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.</td>
</tr>
<tr>
<td>involvement</td>
<td></td>
</tr>
<tr>
<td>3. Incremental</td>
<td>The software is developed in increments with the customer specifying the requirements to be included in each increment.</td>
</tr>
<tr>
<td>delivery</td>
<td></td>
</tr>
<tr>
<td>4. People not</td>
<td>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.</td>
</tr>
<tr>
<td>process</td>
<td></td>
</tr>
</tbody>
</table>
5. Embrace change
Expect the system requirements to change and so design the system to accommodate these changes.

6. 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.

h. Concept of Extreme Programming

Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
1. New versions may be built several times per day;
2. Increments are delivered to customers every 2 weeks;
3. All tests must be run for every build and the build is only accepted if tests run successfully.
4. The extreme programming release cycle

![Figure 1.11]

5. Extreme programming practices

<table>
<thead>
<tr>
<th>Principle or practice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Incremental planning</td>
<td>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.</td>
</tr>
<tr>
<td>2. Small releases</td>
<td>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.</td>
</tr>
</tbody>
</table>
4. Simple design

   Enough design is carried out to meet the current requirements and no more.

5. 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.

6. Refactoring

   Developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.