SOFTWARE ENGINEERING

Tutorial Sheet 2

Submitted By:

Name: Gursimar Kaur

Roll no: 1820036 (Uni.)

Class: D_3 CS E_3

Submitted To:

Prof. Jasdeep Kaur

Q1. What do you mean by the "99% complete" syndrome in software development? Why does it occur? What is its implication for project management? What are its remedies?

Ans1:

Last 1% takes longer than the completed 99%. There are plenty of examples to support this statement.

Example: Consider the progress bars we see on computers, the last period last minute, last minute of a ball match etc. The 90+ percent it runs fast and rest is sluggish. In software development if you do not follow SDLC (Software Development Life Cycle) 99% complete syndrome happens. If you ask developer about the progress he/she may say development is in final stage of just 1% is left .Most probably, this last 1% will drag on to days, weeks, and even few months. Either something was wrong in the SDLC or they failed to follow up. Developers should evaluate the amount of effort needed to complete remaining tasks in hour. Usually instead of this people tend to consider the amount of work gone through against the original estimate.

Q2. Explain the software requirement analysis and specification. Discuss various methods for Requirement gathering.

Ans2:

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.

- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Requirements elicitation/ Requirement Gathering

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users and the customer involved.

1. Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.

Interviews maybe be open ended or structured.

2. Brainstorming Sessions:

It is a group technique and is intended to generate lots of new ideas hence providing a platform to share views. A highly trained facilitator is required to handle group bias and group conflicts. Every idea is documented so that everyone can see it. Finally a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified –

- Normal requirements – In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results etc
- Expected requirements – These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorised access.
- Exciting requirements – It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when an unauthorised access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

Identify all the stakeholders, e.g. Users, developers, customers etc

List out all requirements from customer.

A value indicating degree of importance is assigned to each requirement.

In the end the final list of requirements is categorised as –

It is possible to achieve

It should be deferred and the reason for it

It is impossible to achieve and should be dropped off

5. Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system.

The components of the use case deign includes three major things – Actor, Use cases, use case diagram.

1. Actor – It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
   - Primary actors – It requires assistance from the system to achieve a goal.
   - Secondary actor – It is an actor from which the system needs assistance.
2. Use cases – They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. Use case diagram – A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case.

Q3. Explain the need of software life cycle models in Software Engineering.

Ans 3: Need of SDLC:

1. SDLC is important because it breaks down the entire life cycle of software development thus make is easier to evaluate each part of software development and also makes it easier for programmers to work concurrently on each phase.
2. To have consistencies in deliverables.
3. To increase productivity by executing the project in systematic manner.
4. To reduce the rework effort during project execution.

5. To execute projects with proven framework.
6. To define and focus roles and responsibilities.
7. To enforce plan and work.

Q4. What are the advantages and disadvantages of Waterfall Model of software life cycle?

Ans 4:

Advantages	Dis-Advantages
• Before the next phase of development, each phase must be completed	• Error can be fixed only during the phase
• Suited for smaller projects where requirements are well defined	• It is not desirable for complex project where requirement changes frequently
• They should perform quality assurance test (Verification and Validation) before completing each stage	• Testing period comes quite late in the developmental process
• Elaborate documentation is done at every phase of the software's development cycle	• Documentation occupies a lot of time of developers and testers
• Project is completely dependent on project team with minimum client intervention	• Clients valuable feedback cannot be included with ongoing development phase
• Any changes in software is made during the process of the development	• Small changes or errors that arise in the completed software may cause a lot of problems

Q5. Write a detailed note on

- Requirement gathering

- Requirement analysis and specification

Ans 5:

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Requirement Analysis

The process of studying and analysing the customer and the user or stakeholder needs to arrive at a definition of software requirements.

Requirement Specification

A document that clearly and precisely describes, each of the essential requirements of the software and the external interfaces.

Each requirement is defined in such a way that its achievement is capable of being objectively verified by a prescribed method; for example inspection, demonstration, analysis, or test.

Q6. Differentiate between functional and non-functional requirements

Ans 6:

Sl.No	Functional Requirement	Non-functional Requirement
1.	Defines all the services or functions required by the customer that must be provided by the system	Defines system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
2.	It describes what the software should do.	It does not describe what the software will do, but how the software will do it.
3.	Related to business. For example: Calculation of order value by Sales Department or gross pay by the Payroll Department	Related to improving the performance of the business. For example: checking the level of security. An operator should be allowed to view only my name and personal identification code.
4.	Functional requirement are easy to test.	Nonfunctional requirements are difficult to test
5.	Related to the individual system features	Related to the system as a whole
6.	Failure to meet the individual functional requirement may degrade the system	Failure to meet a non-functional requirement may make the whole system unusable.

Q7. Write four advantages of using software engineering for development of software.

Ans 7:

Advantages of software engineering:

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

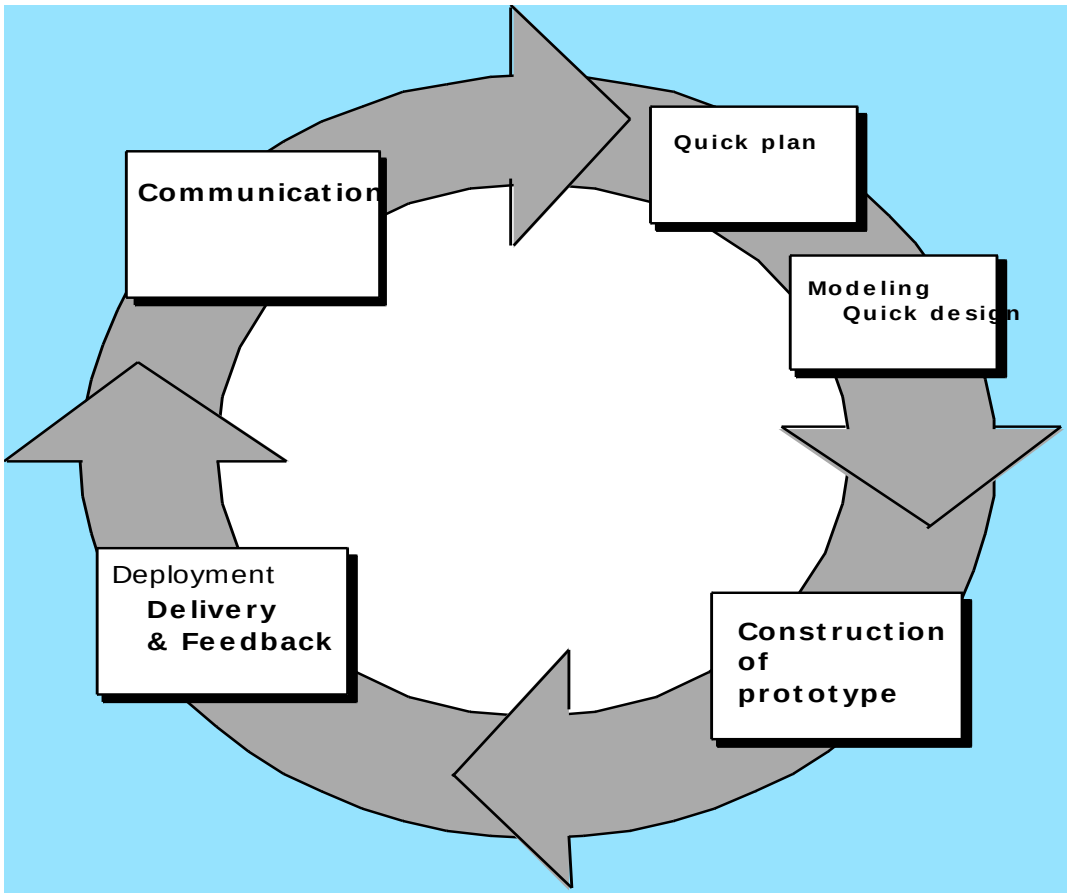
- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

Q8. What is the difference between decision table and decision tree?
Ans 8:

Decision table	Decision tree
■ Helps to clarify the criteria ■ Focuses your attention to framing the problem and reflecting on the different evaluative criteria.	■ Helps to take into account the possible events relevant to your decision. ■ Helps to take into account external factors that might affect your decision. ■ Helps to understand multiple step decisions.

Q9. Explain the following Software life cycle models in detail
(a) Prototyping Model
(b) Spiral Model
Ans 9:
a) Prototyping Model
Customers often define a set of general objectives for Software, but doesn't identify detailed input, processing, or input requirements.
Prototyping paradigm assists the Software engineering and the customer to better understand what is to be built when requirements are fuzzy.



The prototyping paradigm begins with *communication* where requirements and goals of Software are defined.

Prototyping iteration is *planned* quickly and modeling in the form of quick design occurs. The *quick design* focuses on a representation of those aspects of the Software that will be visible to the customer "Human interface". The quick design leads to the *Construction of the Prototype*. The prototype is *deployed* and then *evaluated* by the customer. *Feedback* is used to refine requirements for the Software. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while enabling the developer to better understand what needs to be done. The prototype can serve as the "first system". Both customers and developers like the prototyping paradigm as users get a feel for the actual system, and developers get to build Software immediately. Yet, prototyping can be problematic:

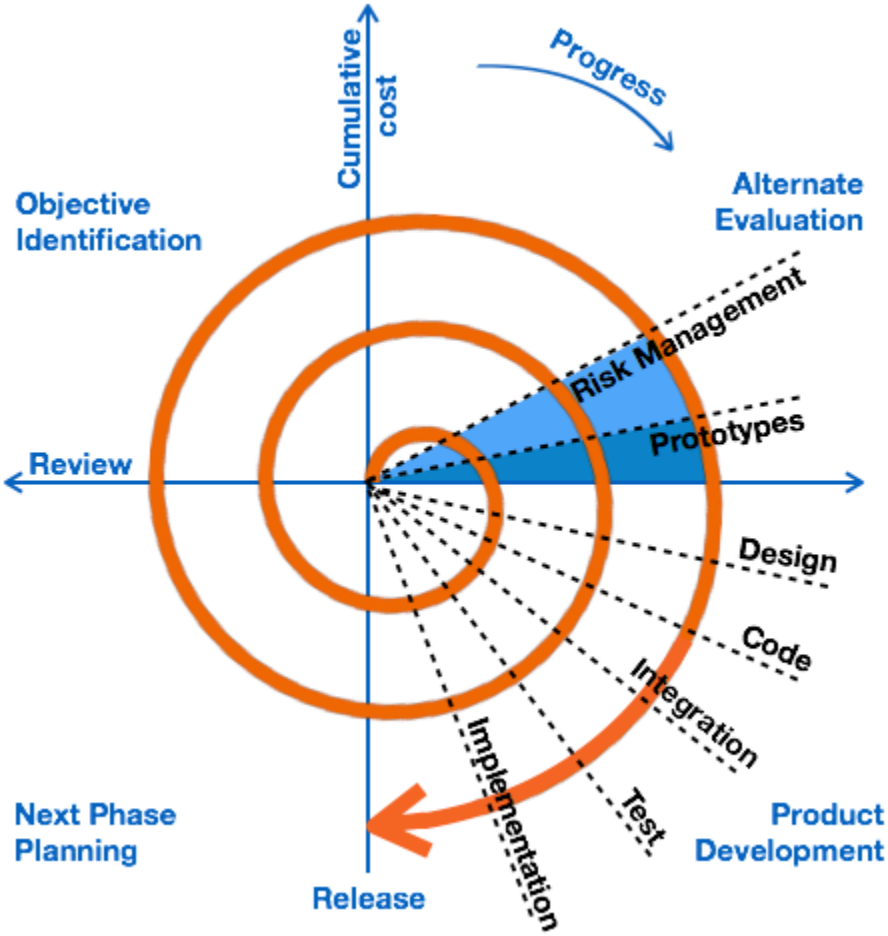
1. The customer sees what appears to be a working version of the Software, unaware that the prototype is held together "with chewing gum.

"Quality, long-term maintainability." When informed that the product is a prototype, the customer cries foul and demands that few fixes be applied to make it a working product. Too often, Software development management relents.

2. The developer makes implementation compromises in order to get a prototype working quickly. An inappropriate O/S or programming language used simply b/c it's available and known. After a time, the developer may become comfortable with these choices and forget all the reasons why they were inappropriate.

b) Spiral Model

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).



This model considers risk, which often goes un-noticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.

Q10. Discuss the relative advantages of formal and informal requirements specifications.

Ans 10:

Advantages of formal requirements specifications:

1. Discovers ambiguity, incompleteness, and inconsistency in the software.
2. Offers defect-free software.
3. Incrementally grows in effective solution after each iteration.
4. This model does not involve high complexity rate.
5. Formal specification language semantics verify self-consistency.

Advantages of informal requirements specifications:

1. Requires little expertise to read or write.
2. Useful for communicating with a broad audience.
3. Useful for capturing intent.

Q11. What are auxiliary functions in algebraic specifications? Why are these needed?

Ans 11.

A considerable part of software systems is comprised of functions that support the main modules, such as array or string manipulation and basic math computation. These auxiliary functions are usually considered less complex, and thus tend to receive less attention from developers. However, failures in these functions might propagate to more critical modules, thereby affecting the system's overall reliability. Auxiliary functions are supportive actions of a software system.

Need of auxiliary functions

While developing auxiliary functions, both agile practices might bring benefits to developers. For instance, compared with solo programmers, more than twice more pair programmers delivered correct implementations, and test-first programming caused the implementation of larger and higher coverage test sets. On the other hand, both agile practices seem to impact negatively on effort in this context. The test-first experiment replication with professionals shows similar results. This evidence shows that, taken The right cautionary measures, the agile practices seem to be a valuable option in the development of auxiliary functionality.