LO1 Describe the Software Life Cycle

LO1.1 Discuss the nature of software and software projects

Introduction

- Developing a good software program is much more than just writing good code.
- Requires thought and planning to ensure that the software works correctly and performs the required functionality.

What is Software (think about languages, hardware, and how hardware/code work together)?

Software:

- Set of instructions (ordered sequence) that tells the computer what to do.
- How a user will interact with the computer hardware.
- Consist of binary values which can change the state of the computer.
- Can be written in machine language, which is the lowest level understandable to specific processors.
- Can be written in a high-level language, such as java, and compiled or interpreted into machine language.

Software Complexity

- Requires designing the software before actually writing the software code.
- Most software requires thought and planning to create efficient, accurate, and effective software.

Types of software

Program

A "Program" is often quickly written by either an individual or a small team to solve a specific task in a specific time and place. They're often specific to the situation in which they are to be used, and not often easily extensible or well tested.

- Designed to be run by the author on the system on which it was developed.

Programming Product

A "Programming Product", as Brooks describes, is a more generalized form of a program. It's more generalized, enabling use by a wider audience and in a wide variety of situations.

Programming product is often the "second generation" of a simple program, as the team has more time to clean up their initial code, making it more robust and widely useful.

- Intended to be run, tested, repaired and extended by anybody
- Thoroughly documented and tested to some proven reliability
- Generalized as much as possible to allow greater utility.

Programming System

Programming systems are collections of individual programs, each implementing a specific task or algorithm, having specific and well defined inputs and outputs. Each program is well documented and tested, with the system as a whole being documented and tested to the same level. Programming systems are often said to implement the "unix philosophy".

- Collection of interacting programs
- Interacting along strictly defined boundaries, with inputs and outputs conforming to precisely defined interfaces.
- Extensively tested in any expected combination of functions.

Programming System Project

Combination of both a programming product and programming system

LO1.2 Discuss the software life cycle stages

Consists of eight distinct stages:

- **Project Initiation**
- **Application(s) Development (boundaries are blurred, overlap, repeat; very organic)**
    - **Analysis**
    - **Design**
    - **Implementation**
    - **Testing**
    - **Installation**
- **Maintenance**
- **Retirement**

LO1.3 Discuss the major activates within each Lifecycle stage

Project Initiation (WHY/WHO/WHERE)

Involves any activities that are required to initiate the project. Includes finalizing details necessary to begin the project: establishing the required functionality, finalizing price and agreement on price.

This includes:

- Acceptance of a project proposal
- Create a project charter
- Create a project plan
- Form the project team
- Obtaining needed equipment or other resources
- Finalization of contracts with the customer

A typical project includes a

Project Proposal:

- created by the customer so that they can prove that they require the project.
- customer may obtain assistance in completing the project proposal.
- proposal should identify the project objective and project scope.

During this stage, a **project plan and/or** a project charter should also be written.

Project Plan:

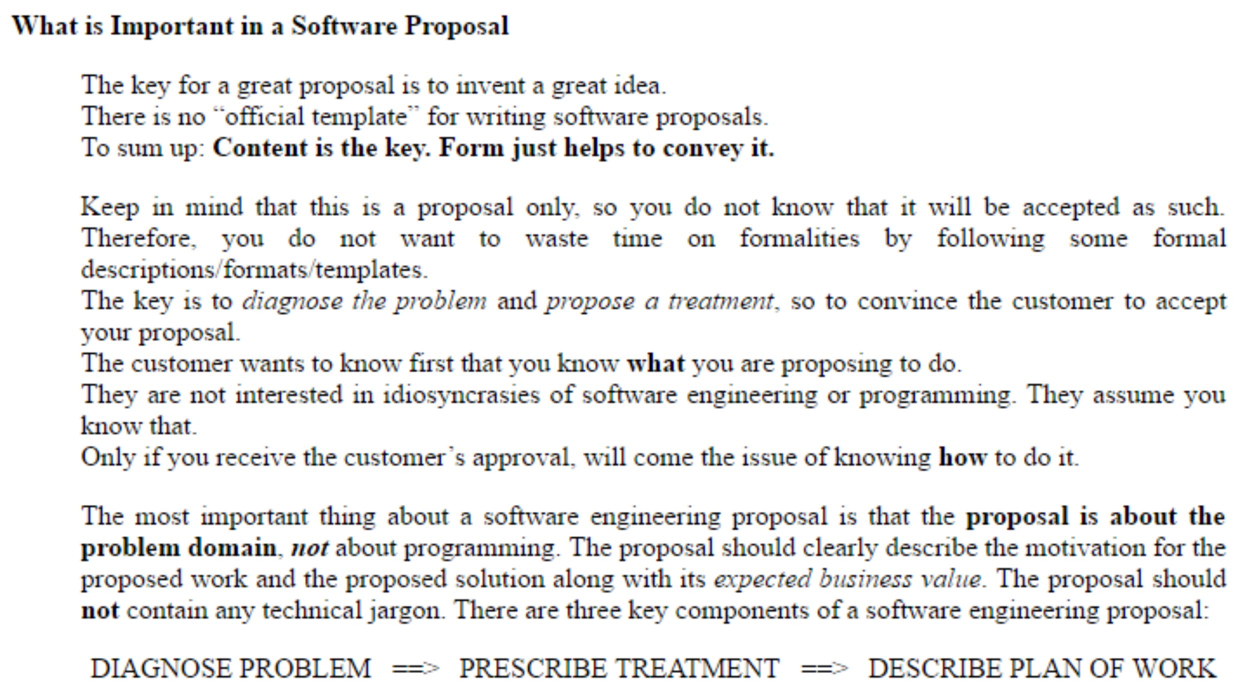
- detailed document of how the project will be accomplished
- written by the development team.
- It includes details of: objectives, scope, resources estimate (people and equipment), complete list of deliverables, risks, project team information and details of standards to be followed.

Project Charter:

- Charter is usually a summary of the project. Think: executive summary of the Project Plan
- Forms the basis of the "agreement" between the developers and the clients
  - If you were going to build a house, the charter would be the agreements you sign with the contractor
  - The blueprints would be the specifications of the building along with the details of how the building will be constructed
- Example: http://isoconsultantpune.com/wp-content/uploads/2015/05/MassCommunicationProjectCharter.pdf

Items handled during the project initiation stage are:

- Project proposal (borrowed from http://www.ece.rutgers.edu/~marsic/Teaching/SE/proposal.html) :



What is Important in a Software Proposal

The key for a great proposal is to invent a great idea.
There is no "official template" for writing software proposals.
To sum up: **Content is the key. Form just helps to convey it.**

Keep in mind that this is a proposal only, so you do not know that it will be accepted as such. Therefore, you do not want to waste time on formalities by following some formal descriptions/formats/templates.
The key is to *diagnose the problem* and *propose a treatment*, so to convince the customer to accept your proposal.
The customer wants to know first that you know **what** you are proposing to do.
They are not interested in idiosyncrasies of software engineering or programming. They assume you know that.
Only if you receive the customer's approval, will come the issue of knowing **how** to do it.

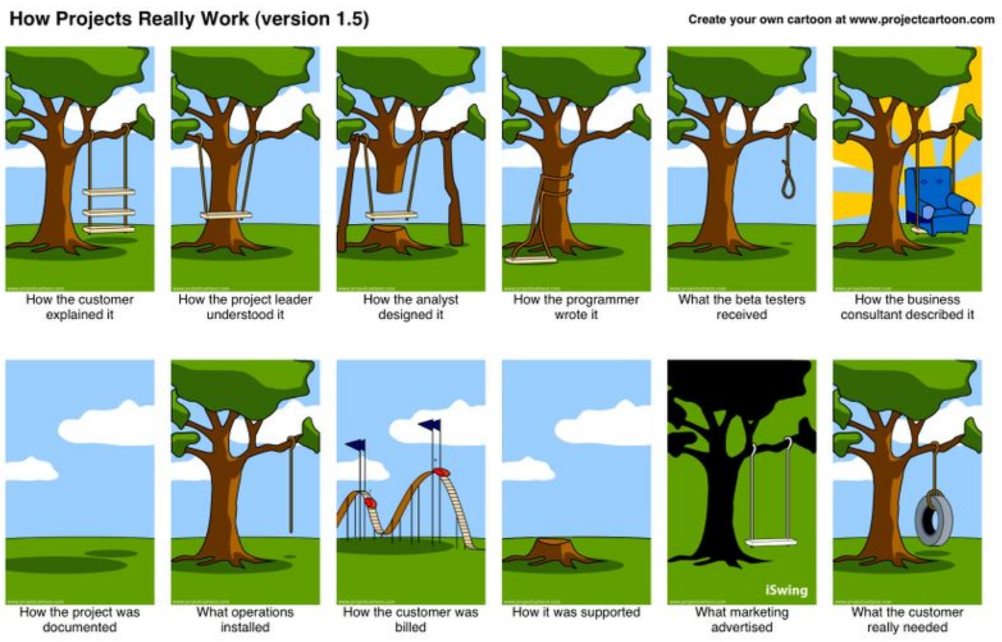
The most important thing about a software engineering proposal is that the **proposal is about the problem domain**, ***not*** about programming. The proposal should clearly describe the motivation for the proposed work and the proposed solution along with its *expected business value*. The proposal should **not** contain any technical jargon. There are three key components of a software engineering proposal:

DIAGNOSE PROBLEM ==> PRESCRIBE TREATMENT ==> DESCRIBE PLAN OF WORK

- Project Charter

Analysis (WHAT)

During this stage, the development team works to finalize the requirements for the project.


How Projects Really Work (version 1.5)
Create your own cartoon at www.projectcartoon.com
How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
What the beta testers received
How the business consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
iSwing
What marketing advertised
What the customer really needed

This provides the details of "**what**" functionality the project needs to provide.

- The project scope is analyzed to determine each detailed requirement of the system. This is accomplished by questioning the client, questioning the employees who will use the system, and studying any documentation currently in use.
- During this stage, the development team will determine what data is needed, when it is needed, and its structure.

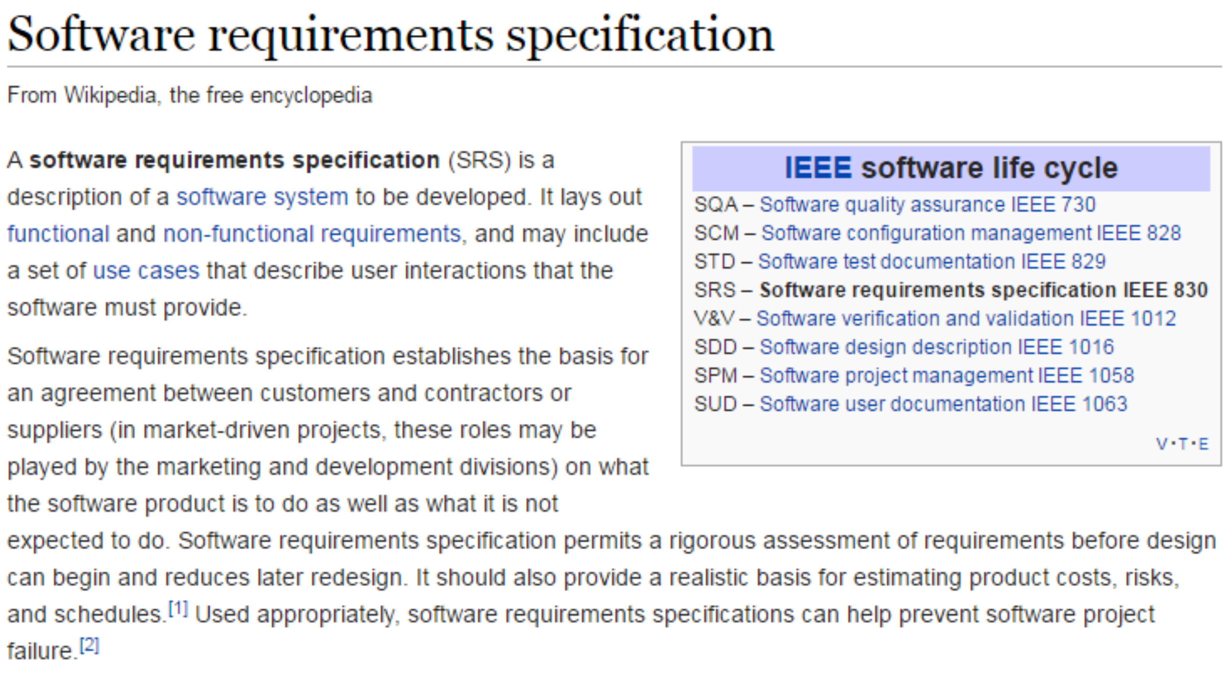
- Areas that are focused on are business rules, processes and data employed at the customer site.

    - **Business rules** – determine the logic that defines the process and is based on the needs of the business. The business rules dictate how things will be done. Ex. A business rule in the CST program would be that the student must receive a mark of 50% in order to pass the course.
    - **Processes** – are a set of specific activities that are followed to produce specific output. Ex. A process may be the act of registering a new student into the database. Processes incorporate business rules. The analysis team should walk the client through each process step-by-step, to ensure that each step of the process is documented correctly.

- **Data** – is the facts and figures used within the system. The data required for the project must be identified. As well, it is important to know if the data is persistent or temporary.

Items created during the Analysis stage

- **Requirements Document**



Software requirements specification

From Wikipedia, the free encyclopedia

A **software requirements specification** (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

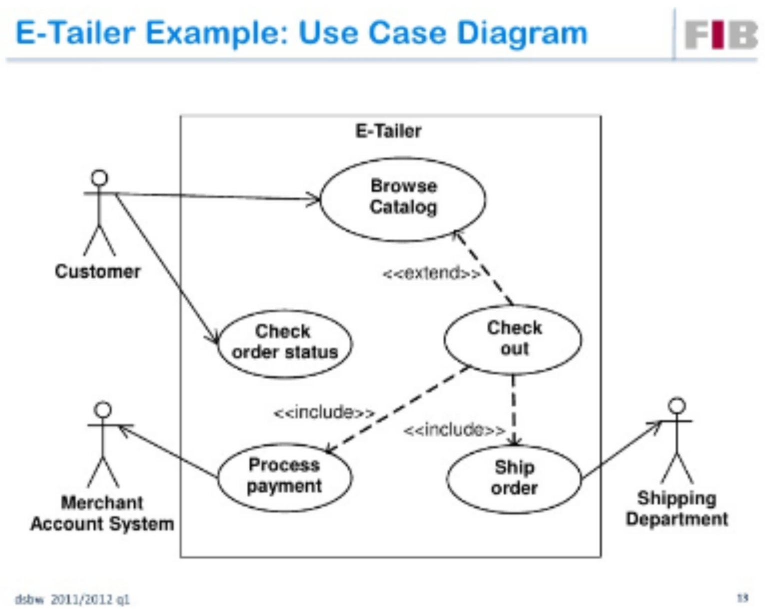
Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.[1] Used appropriately, software requirements specifications can help prevent software project failure.[2]

IEEE software life cycle

SQA – Software quality assurance IEEE 730
SCM – Software configuration management IEEE 828
STD – Software test documentation IEEE 829
SRS – **Software requirements specification IEEE 830**
V&V – Software verification and validation IEEE 1012
SDD – Software design description IEEE 1016
SPM – Software project management IEEE 1058
SUD – Software user documentation IEEE 1063

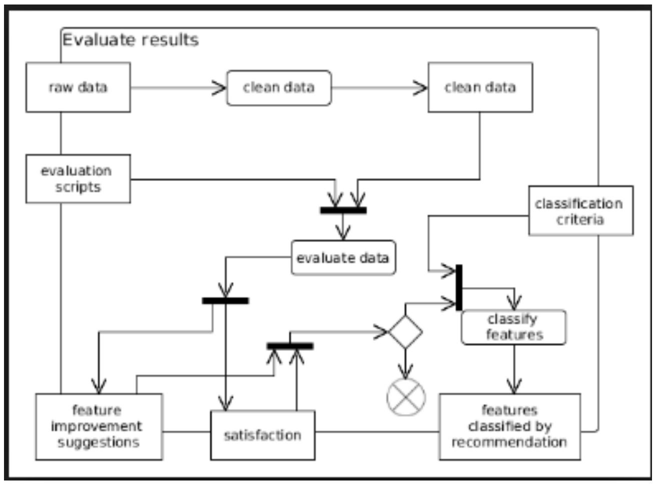
V · T · E

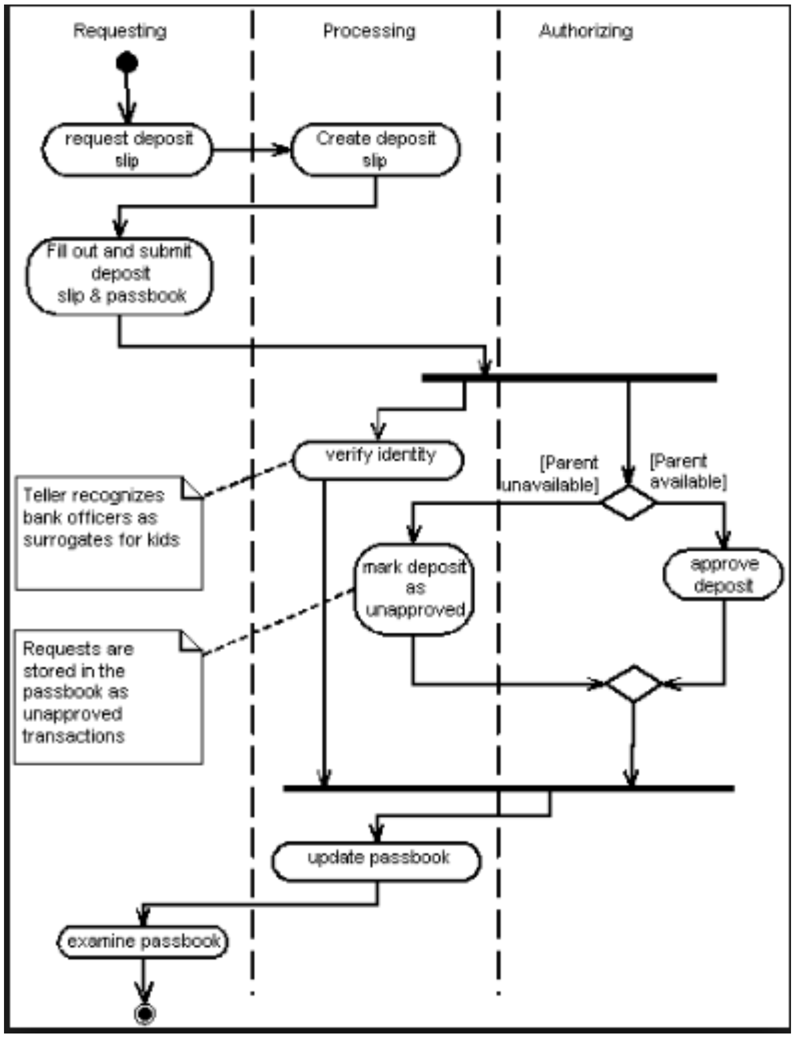
- **UML Use Case Model (includes stories)**



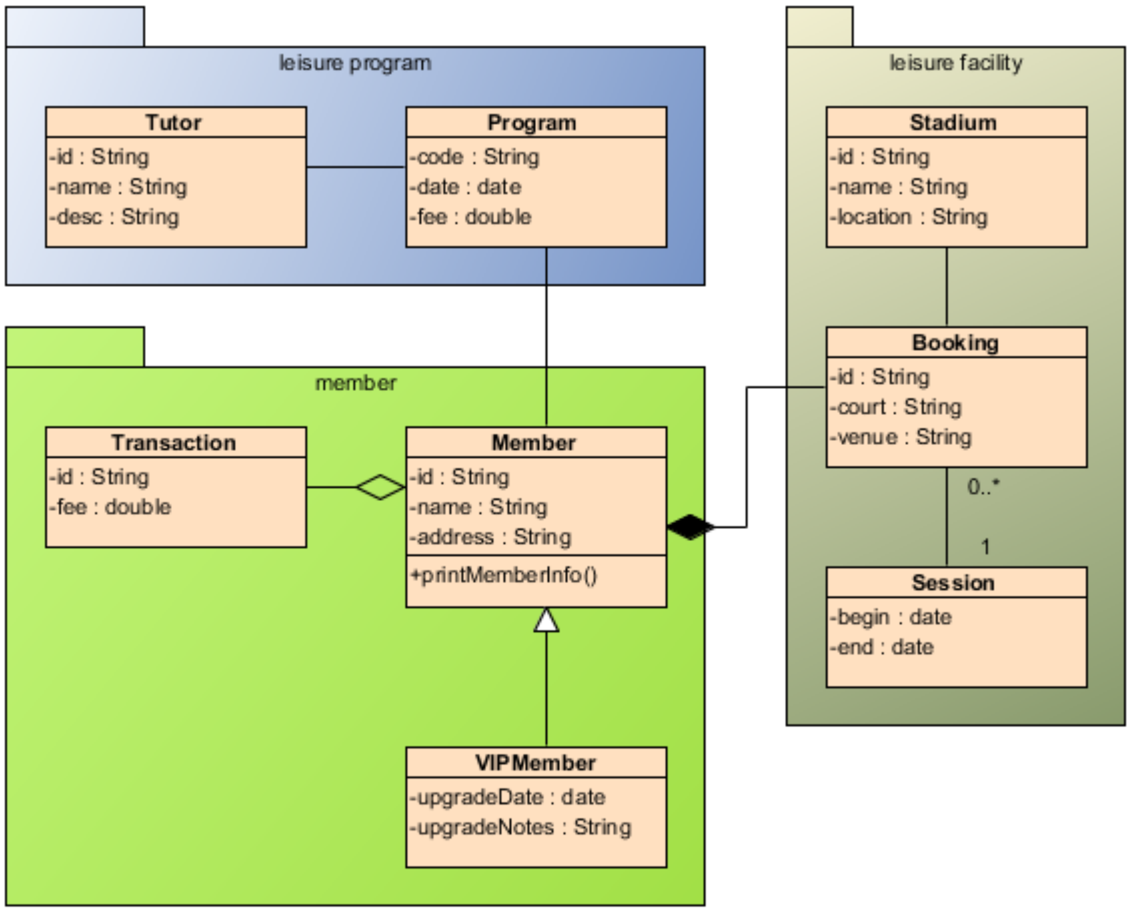
E-Tailer Example: Use Case Diagram

- **UML Activity Model**




Requesting
Processing
Authorizing
request deposit slip
Create deposit slip
Fill out and submit deposit slip & passbook
verify identity
Teller recognizes bank officers as surrogates for kids
[Parent unavailable]
[Parent available]
mark deposit as unapproved
approve deposit
Requests are stored in the passbook as unapproved transactions
update passbook
examine passbook

- **UML Analysis Classification Model**



- **Acceptance Test** for each business function.
  - **What are they**
  - **When do we capture them**
  - **How are they written**
  - **Who executes them**
  - **Where do we execute them**

Design (HOW)

Based on the requirements, a more detailed plan of how to develop the software is determined. The design must meet all the functional and data requirements for the project. This is the “**how**” stage.

- In this stage a blueprint for the hardware and software is developed based on the requirements of the system.
- Development environment is finalized.
- Logical requirements are translated into algorithms or solution strategies.
- The data requirements refined by specifying data type and size.
- Persistent storage structures are detailed.

Items created and modified during the design stage:

- **UML Implementation Classification Model**
- **Revised UML Use Case Model**
- **Revised UML Activity Model**
- **UML Sequence Model**
- **UML State Change Model**
- **Database Schema**

Implementation (DO)

The development team takes the detailed plan created in the design stage and creates code based on the plan. This stage is when the actual writing of the software is performed. It is often done at the same time as the Testing stage.

- Software design is translated into actual software.
- Code is written for the first time.
- Database queries are created and initial data values are setup.
- The software and hardware are configured based on the architectural requirements.

Items handled/created during this stage:

- **Revised UML Implementation Classification Model**
- **Code**
- **Database**
- **Additional files – like scripts, configurations files, etc.**
- **Installation software**

Testing (Verification and Validation)

This stage is the verification of the software that was written. This stage ensures that the software was written to accomplish the functionality it was intended to (which is determined in the analysis stage), and ensures that the software was written (during the implementation stage) without error.

- System is evaluated for correctness
- Ensuring the system meets the requirements and operates in a correct manner.

Items handled during this stage:

- **White box testing** (unit testing) – this technique where the tester has access to the internals of the code being tested. Purpose: to ensure the software's implementation matches the design
- **Black box testing** (functional testing or acceptance test) – this technique is where the test does not have access to the internals of the code being tested. Tester focuses on the interfaces to the system. Purpose: to ensure the software satisfies the requirements during the Analysis stage

Items handled during this stage:

- **White box test results**
- **Black box test results**

TDD (Test driven development) combines continuous testing with code generation by setting the tests first and then coding towards passing the tests.

Validation is obtained via Acceptance Tests (performed by client).

Installation

Installation of the software at client site. The system must be initialized and brought to a correct running state.

- Includes the actual placing of software, database, and web content and support software onto the client computer.
- During installation, the development team will configure and initialize the database, application software, complete a test run, deliver user training.

Items handled during this stage

- Create user and installation manual

Maintenance

During this stage changes are made to the software to support additional functionality, or to fix errors that exist in the system. Surprisingly, this stage typically lasts longer than any other stages, and accounts for the majority of the cost of most systems.

- Providing support for the system after it has been installed.
- The longest and most expensive stage of the cycle
- Includes fixing errors that were not detected earlier in the project.
- Additional or modified functionality may be required to support changing system requirements.

Retirement

This stage is where the system ceases to be used

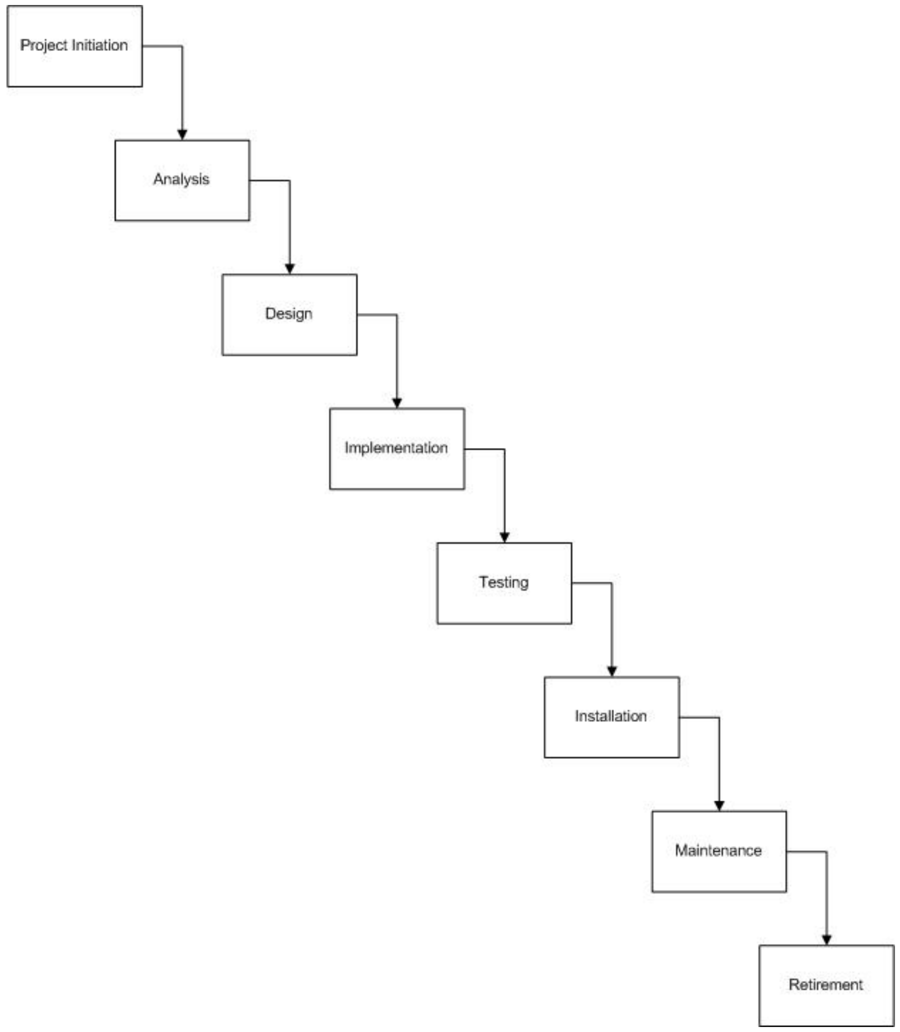
- Occurs when the system is no longer to be used.
- The development team shuts down the system, and then decommissions it.
- The final act of the development team is to remove all sensitive data from the hardware.

LO1.4 Discuss common software development methodologies

Software Models

The software life cycle stages can be followed in different manners, including a number of different models. The discussed models are as follows:

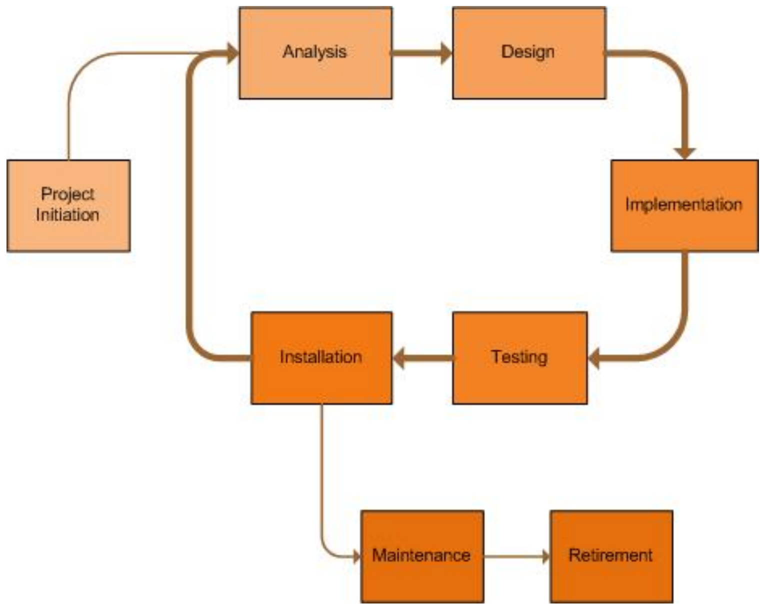
Waterfall



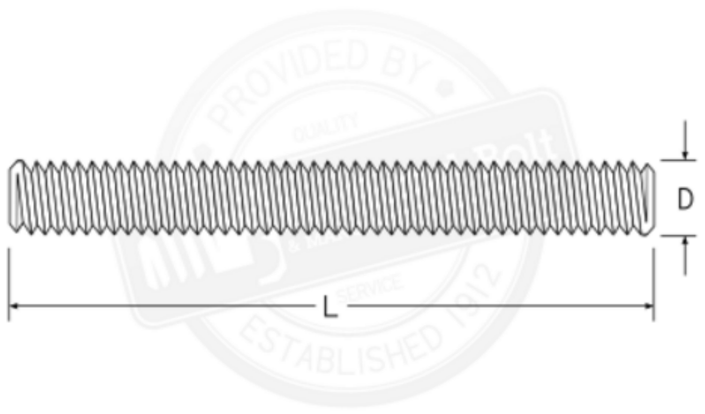
This is the traditional view of software development. It was commonly used in the early days of software development. Each stage is completed in its entirety before moving on to the next stage. It isn’t used so much any longer because it is the most ridged and doesn’t allow for changing things part way through.

A linear process like waterfall, where each step flows only forwards into the next, tends to not work well with projects where requirements are likely to change (characteristics of the project that might indicate this may happen?). While this type of approach might work well for engineering or physical structure projects like bridges or roads, where major requirements changes are unlikely once the project is underway, it tends to not deal well with software projects, where changes are almost a constant factor.

Iterative



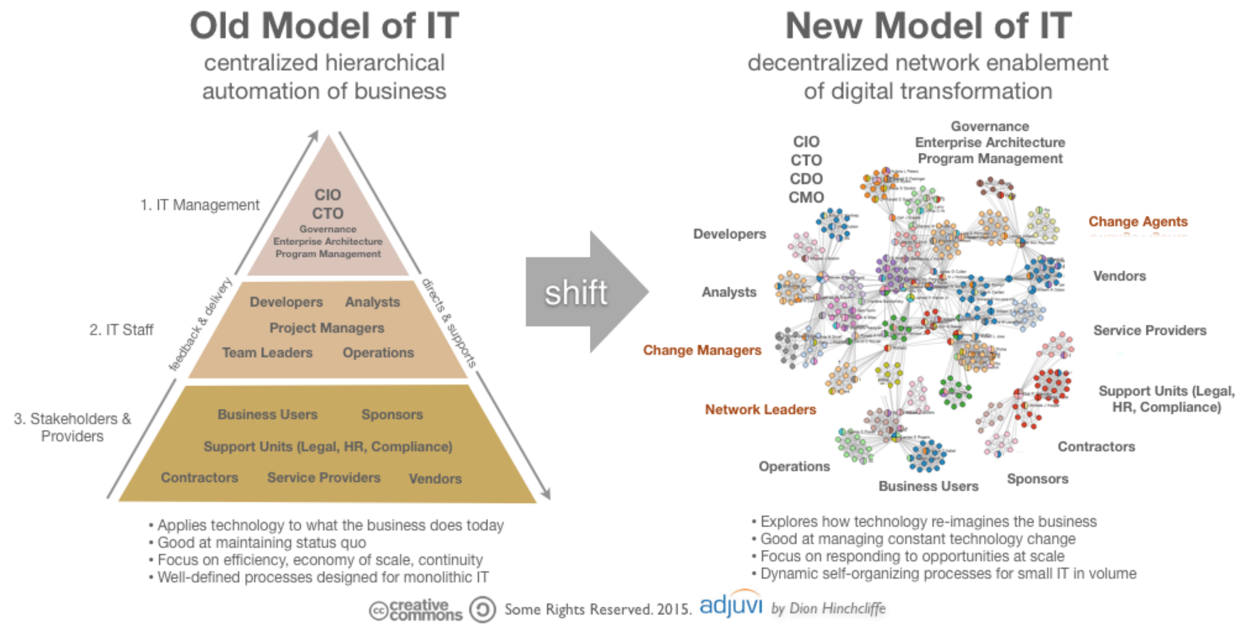
The Iterative or incremental model is a cyclic approach to software development. It was introduced to deal with the weaknesses of the waterfall model. In Iterative development, parts of the system are developed at different times. The system is built up in parts so that a running system with limited functionality is achieved earlier in the schedule. Functionality is added to the system throughout the schedule until the project is complete. The development of the software still involves analysis, design, implementation and testing, but it is done in increments. Progress on the project resembles the threads on a bolt:



The length of the project is represented by L; D represents the length of a cycle through the iteration which ideally will be short: two weeks to one month at most.

Some form of the iterative model is used for most software development today. This approach offers good flexibility. It reduces risk by developing a running system with minimal functionality early in the project. Mistakes made developing early parts of the project can be corrected later before software is fully developed.

Evolution: Changing the Software Development Model



- Applies technology to what the business does today
- Good at maintaining status quo
- Focus on efficiency, economy of scale, continuity
- Well-defined processes designed for monolithic IT

- Explores how technology re-imagines the business
- Good at managing constant technology change
- Focus on responding to opportunities at scale
- Dynamic self-organizing processes for small IT in volume

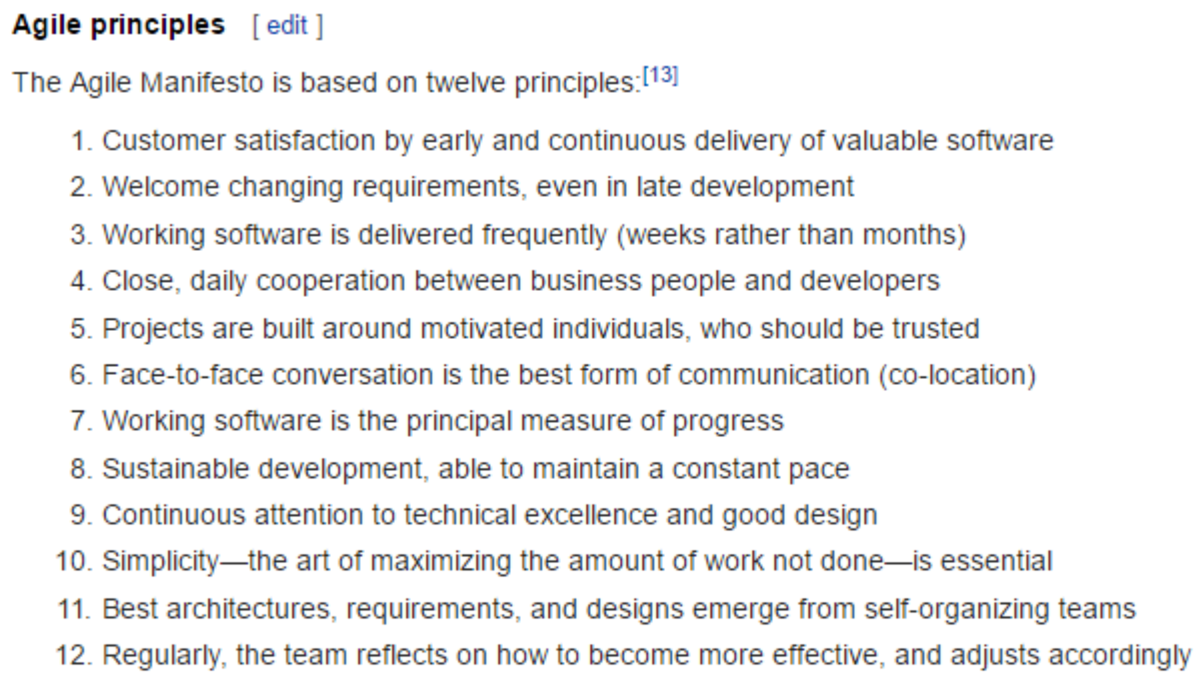
Agile

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

Agile software development is methodology that encompasses more than just the approach to the development of the software. At its base agile methodology uses an iterative approach to software development so it goes through the same life cycle stages already outlined, but goes beyond that to promote open collaboration and process adaptability throughout the life cycle of the project. The frequency of the feedback loops used in iterative development is dramatically sped up, sometimes to the point where teams create builds of their software multiple times per day, releasing to the client for testing every week or so.

Agile recommends an evolutionary approach to software design, where rather than attempting to fix a class/method design at the beginning of a project during the analysis and design stages, as much of the design as possible is moved to later in the project on an as-needed basis. Since project teams are likely to learn more about their client's needs once the client has had a chance to use their software, they can reduce the impact of change by having done less work up front.

The Agile Manifesto: http://agilemanifesto.org/



Agile principles [edit]

The Agile Manifesto is based on twelve principles:[13]

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile promotes client or client representative on site, total involvement of the client from the start, much more open, less formal approach. It focuses more on the project team and their interactions, giving the developers more control over their situation and the progress of the software project. This is in stark contrast to a more plan-driven approach like waterfall, where developers and other team members are "slotted in" to particular places or roles within the project.

TDD is considered part of Agile. Agile also contains the following approaches: