CHAPTER 2
LITERATURE REVIEW

2.1. Network Definition

According to Tanenbaum (2011) and Forouzan, (2007), computer network is defined as nodes that are interconnected in the same network and could share information.

In order to be called network, it has to have some important criteria, there are:

a. Performance

   Performance is measured by looking at its transit time and response time.

b. Reliability

   Reliability is measured by looking at the rate of failure, recover time from a failure, and the network's strength in handling problems.

c. Security

   Security is measured by how safe the data from breaches, data loss, data damage, and unauthorized access.

2.1.1. Broadcasting

According to Fall and Stevens (2012:436-436), broadcasting is a method to send a message to all possible receivers in a network. The router only forwards a copy the message it receives to all of interface other than the message sender.

2.1.2. Multicasting

Multicasting is a method of sending traffic only to those receivers that are interested in the message. The receiver was chosen by indicating the receivers or instead having the receivers independently indicate interest in the message. The network began sending traffic to the specified recipient after it has received a message. (Fall and Stevens, 2012:436-436)

2.2. Type of Connection

2.2.1. Point-to-Point

Point-to-point connection is a directed link between two nodes. The connection is established to send messages called *packets.*

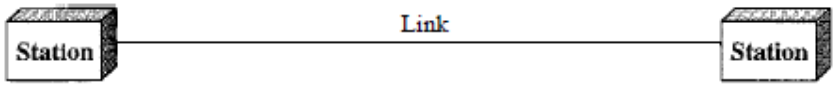

Figure 2.1 - Point-to-point Connection (Source: Forouzan, 2007:9)

2.2.2. Multipoint

Multipoint connection or multidrop connection is a connection which connects two or more devices in a single link.

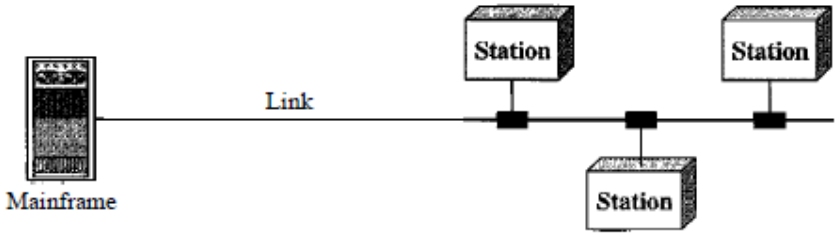

Figure 2.2 - Multipoint Connection (Source: Forouzan, 2007:9)

2.3. Categories of Network

Tanenbaum (2011:18-24) stated that network categories are separated by its coverage area size, there are:

2.3.1. Personal Area Network (PAN)

PAN is a connection for the devices to communicate only at the range of a person. This connection could be established by cables or wireless connection. Example of PAN is a computer connected with its peripherals through some network.

2.3.2. Local Area Network (LAN)

LAN usually covers all devices connected in a single building or office. LAN connection area is only a few kilometers. Due to its small coverage area, LAN usually uses only one transmission medium and topology type. The topologies are bus, ring, and star topology.

2.3.3. Metropolitan Area Network (MAN)

MAN has a size bigger than LAN and smaller than WAN. It covers the area inside a city and usually has endpoints throughout the city. Cable television is the example of MAN.

2.3.4. Wide Area Network (WAN)

WAN has the biggest coverage area compared to the other two. It could cover a whole country and even the entire world.

Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Figure 2.3 - Connection Classification (Source: Tanenbaum, 2011:18)

2.4. Transmission Type

This project is able to establish connection using either wired or wireless connection.

2.4.1. Wired Connection

Wired connection established its connection through transmission media, such as copper wire and fiber optics. Wired connection is divided into two types, which is *classic Ethernet* and *switched Ethernet.* Classic Ethernet is the original form of Ethernet that runs at 3 – 10Mbps.

Switched Ethernet is the advanced form of Ethernet that runs at 100Mbps (Fast Ethernet), 1,000Mbps (Gigabit Ethernet), 10,000Mbps (10 Gigabit Ethernet). Practically, only switched Ethernet is used in these days. (Tanenbaum, 2011:280).

2.4.2. Wireless Connection

Wireless LANs are the newest evolution in LAN technology. Wireless Connection is an unguided media transport electromagnetic waves without using a physical conductor. Signals are normally broadcast through free space and thus are available to anyone who has a device capable of receiving them. (Forouzan, 2007:421).

2.5. TCP/IP Architecture

The TCP/IP protocol suite was developed prior to the OSI model. Therefore, the layers in the TCP/IP protocol suite do not exactly match those in the OSI model. The original TCP/IP protocol suite was defined as having four layers: host-to-network, internet, transport, and application. However, when TCP/IP is compared to OSI, we can say that the host-to-network layer is equivalent to the combination of the physical and data link layers. The internet layer is equivalent to the network layer, and the application layer is roughly doing the job of the session, presentation, and application layers with the transport layer in TCP/IP taking care of part of the duties of the session layer. TCP/IP protocol suite is made of five layers: physical, data link, network, transport, and application. The first four layers provide physical standards, network interfaces, internetworking, and transport functions that correspond to the first four layers of the OSI model. The three topmost layers in the OSI model, however, are represented in TCP/IP by a single layer called the application layer. (Forouzan, 2007:42).

2.5.1. Physical and Data Link Layer

According to Forouzan (2007:43-44), at the physical and data link layers, TCP/IP does not define any specific protocol. It supports all the standard and proprietary protocols. A network in a TCP/IP internetwork can be a local-area network (LAN) or a wide-area network (WAN).

2.5.2. Network Layer

The network layer, TCP/IP supports the Internetworking Protocol. IP uses four supporting protocols: ARP, RARP, ICMP, and IGMP.

2.5.2.1. Internetworking Protocol (IP)

Internetworking Protocol (IP) is the transmission mechanism used by the TCP/IP protocols. It is unreliable and connectionless protocol-a-best-effort delivery service. Best effort in these term means that IP provides no error in checking or tracking.

IP transports data in packets called datagrams, which transported separately between each datagrams. IP does not keep track of the routes and has no facility for reordering datagrams once they arrive at the destination.

2.5.2.2. Address Resolution Protocol (ARP)

The Address Resolution Protocol (ARP) is used for connecting a logical address with a physical address. On a physical network, in example LAN, each device are identified by a physical address, usually imprinted on the Network Interface Card (NIC). ARP is used to find the physical address of a device when its Internet Address is known.

2.5.2.3. Reverse Address Resolution Protocol (RARP)

The Reverse Address Resolution Protocol (RARP) allows a host to discover the Internet address when it knows only its physical address. It is used when a computer is

connected to a network for the first time or when a diskless computer is booted.

2.5.2.4. Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol is a method used by hosts to send notification of datagram problems back to the sender. ICMP messages are query and error reporting messages.

2.5.2.5. Internet Group Message Protocol (IGMP)

The Internet Group Message Protocol (IGMP) is used to facilitate the simultaneous transmission of a message to a group of recipients.

2.5.3. Transport Layer

This project will use UDP as its transport protocol. UDP allow the usage of broadcast method that will be used in peer discovery. TCP as a transport protocol cannot use broadcast.

According to Forouzan (2007:44-45), the transport layer was represented by two protocols, which are TCP and UDP. IP is a host-to-host protocol, which means that it can deliver packets from one physical device to another device. UDP and TCP itself are transport level protocols that deliver a message from a process to another process.

2.5.3.1. User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a process-to-process protocol that adds only port addresses, checksum error control, and length information to the data from the upper layer.

Forouzan (2007:709) called UDP as a connectionless, unreliable transport protocol. It only provides process-to-process communication instead of host-to-host communication to the services of IP. In addition to its disadvantages, UDP comes with good advantages also. UDP

is a very simple protocol that uses a minimum of overhead. If a process wants to send a small message and does not care much about reliability, ut can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver that using TCP.

UDP shared the same concepts to the transport layer, which are:

a. Connectionless Services

Connectionless services means that each user datagram sent by UDP is an independent datagram. Even if the datagram comes from the same source process and going to the same destination program, there is no relationship between each datagram.

There is no connection establishment and no connection termination, which means that each user datagram can travel on a different path.

b. Flow and Error Control

There is no flow control in UDP. Therefore, UDP has no window mechanism. The recipient may overflow with incoming message.

There is no error control mechanism except for the checksum in UDP, which means that the sender does not know if the message has been lost or duplicated. When the recipient detects an error through the checksum, the datagram is discarded without anyone knowing.

c. Encapsulation and Decapsulation

UDP protocol uses encapsulation and decapsulation to send a message in an IP datagram.

d. Queuing

A process can only obtain one port number and one outgoing queue and one incoming queue. The queue will be functioning until the process terminates.

Client process can send messages to the outgoing queue by using the port number specified in the request. After there is message on the queue, UDP removes it one by one after it has been added to the UDP header and delivered it to IP. Operating system can ask the client process to wait before sending any more messages if the outgoing queue overflowed.

When client receives message, UDP will check to see if an incoming queue has been created for the port number specified. If there is a queue for it, UDP sends the received user datagram to the end of the queue. If there is no queue for it, UDP discards the received user datagram and uses ICMP to send a port unreachable message to the server. All of the incoming message, whether it comes from different server, it will go to the same queue. UDP will drop the user datagram if the incoming queue overflowed asks to send port unreachable message to the server.

According to Fall and Stevens (2012:435), UDP transport protocol take advantage of broadcasting and multicasting, allowing an application to send a single message to multiple recipients.

2.5.3.2. Transmission Control Protocol (TCP)

According to Forouzan (2007:45), the Transmission Control Protocol (TCP) provides full transport-layer services to applications. TCP is a reliable stream transport protocol. Stream in this context means connection-oriented. In TCP, a connection between both devices must be created first before it can transmit data.

TCP divides a stream of data into smaller units called segments at the sending of each transmission. Each of it has a number used for reordering the segments. At the receiving end, TCP collects all the datagram and reorder it based on sequence numbers.

Fall and Stevens (2012:435) stated that TCP implies a connection between two hosts and one process on each host. TCP can use unicast and anycast addresses, but TCP cannot use broadcast or multicast addresses.

2.5.4. Application Layer

The application layer is used for the user to access the network using user interface to interact.

2.6. C++ Programming Language

This project uses C++ programming language since 3DS MAX SDK uses C++ as its programming language.

According to Stroustrup (2013:11-38), C++ is a compiled language. In order to run a program, its source code has to be processed by a compiler, producing object files, which are combined by a linker yielding an executable program. One C++ program usually contain of many source code file. An executable program is created for a specific hardware/system, for example a Windows application is not suitable for Mac OS. The C++ language features most directly support four programming styles, which are:

2.6.1. Procedural Programming

This type of programming focused on processing and the design of suitable data structures. C++ provides more supports for procedural programming in the form of many additional language constructs and a stricter, more flexible, and more supportive type system.

2.6.2. Data Abstraction

This type of programming focused on the design of interfaces, hiding implementation details in general and representations in particular. C++ supports concrete and abstract classes.

2.6.3. Object-oriented Programming

This type of programming focused on the design, implementation, and use of class hierarchies. C++ provides simplifying the definition of a class out of existing ones. Class hierarchies provide run-time polymorphism and encapsulation.

2.6.4. Generic Programming

This type of programming focused on the design, implementation, and use of general algorithms. General algorithm means that the algorithm can be designed to accept a wide variety of types as long as they meet the requirements on its arguments.

2.7. Windows Socket (Winsock)

This project uses WinSock as its main connection, since the project uses Windows Operating System as the base.

According to Russinovich et al. (2012), the original Windows Sockets (Winsock) version 1.0 was Microsoft's implementation of BSD (Berkeley Software Distribution) Sockets, a programming API that became the standard by which UNIX systems have communicated over the Internet since the 1980s. The modern versions of Winsock include most of the functionality of BSD sockets but also include Microsoft-specific enhancements, which continue to evolve. Winsock supports reliable, connection-oriented communications as well as unreliable, connectionless communication. Reliable connection

indicates whether the sender is notified of any problems in the delivery of data to the receiver.

WinSock system can operate over TCP/IP-based wide area networks, and the server, for instance, can accept requests from UNIX, Linux, and other non-Windows clients. (Hart, 2010:411-412).

Winsock, by enabling standards-based interoperability, allows programmers to exploit higher-level protocols and applications, such as ftp, http, RPCs, and COM, all of which provide different, and higher-level, models for standard, interoperable, networked interprocess communication. The Winsock API was developed as an extension of the Berkeley Sockets API into the Windows environment, and all Windows versions support Winsock. Winsock's benefits include the following:

- Porting code already written for Berkeley Sockets is straightforward.
- Windows machines easily integrate into TCP/IP networks, both IPv4 and IPv6. IPv6, among other features, allows for longer IP addresses, overcoming the 4-byte address limit of IPv4.
- Sockets can be used with Windows overlapped I/O, which, among other things, allows servers to scale when there is a large number of active clients.
- Sockets can be treated as file HANDLEs for use with ReadFile, WriteFile, and, with some limitations, other Windows functions, just as UNIX allows sockets to be used as file descriptors. This capability is convenient whenever there is a need to use asynchronous I/O and I/O completion ports.
- Windows provides non-portable extensions.
- Sockets can support protocols other than TCP/IP, but this chapter assumes TCP/IP.

A newer version of Windows Sockets, which is Windows Socket 2, adds several areas of functionality (Hart, 2011:448):

- Standardized support for overlapped I/O which is the most important enhancement.
- Scatter/gather I/O (sending and receiving from noncontiguous buffers in memory).

- The ability to request quality of service (speed and reliability of transmission).
- The ability to organize sockets into group. The quality-of-service of a socket group can be configured, so it does not have to be done on a socket-by-socket basis. Also, the sockets belonging to a group can be prioritized.
- Piggybacking of data onto connection requests.
- Multipoint connections (comparable to conference calls).

2.8. Distributed Rendering

Sheharyar and Bouhali (2013:117) states that rendering is compute-intensive and time-consuming process, but it is a highly parallelizable task.

2.8.1. Types of Parallelism

According to Crockett (1997:1-10), parallel rendering has been applied to virtually every image generation technique used in computer graphics, including surface and polygon rendering, terrain rendering, volume rendering, ray-tracing, and radiosity.

There are several types of parallelism that can be applied in the rendering process, which are:

2.8.1.1. Functional Parallelism

Functional parallelism method is to split the rendering process into several distinct functions which can be applied in series to individual data items. If a processing unit is assigned to each function and a data path is provided from one unit to the next, a rendering pipeline is formed. After each item has been completed by processing unit, it forwards the item to the next unit, and receives a new item to be completed.

Functional approach has been mapped very successfully into the special-purpose rendering hardware used in a variety of commercial computer graphics workstations.

Functional approach also has two significant limitations.

a. The overall speed of the pipeline is limited by its slowest stage. Therefore the functional units must be designed carefully to avoid bottlenecks.
b. The available parallelism is limited to the number of stages in the pipeline.

2.8.1.2. Data Parallelism

Data parallelism method is to split the data into multiple streams and operate on several items simultaneously by replicating a number of identical rendering units.

The data-parallel approach can take advantage of large numbers of processors and has been adopted in one form or another by most of the software renderers. Data parallelism also lends itself to scalable implementations, allowing the number of processing elements to be varied depending on factors such as scene complexity, image resolution, or desired performance levels.

Two classes of data parallelism which can be identified in rendering process are object parallelism and image parallelism.

a. Object Parallelism

   Object parallelism creates the first few stages of the rendering pipeline including modeling and viewing transformations, lighting computations, and clipping.

b. Image Parallelism

   Image parallelism occurs in the later stages of the rendering pipeline and includes the operations used to compute individual pixel values. Pixel computations include illumination, interpolation, composition, and visibility determination.

There is a strategy in partitioning an image (image partitioning), which is block of contiguous scanlines, by dividing the image into a large blocks of contiguous pixels as shown on Figure 2.4.

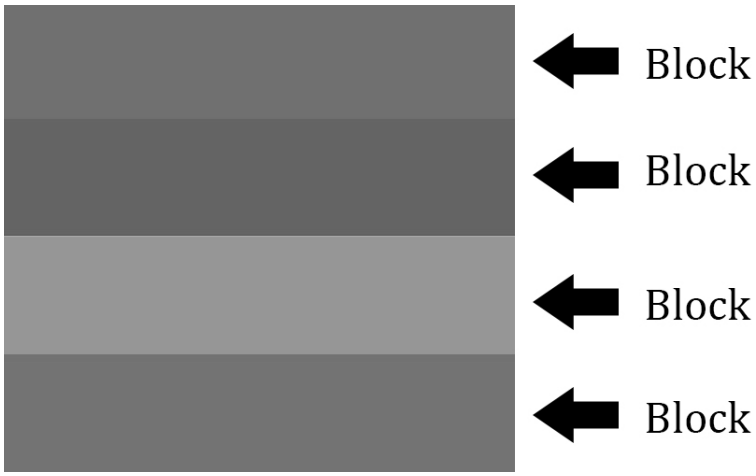

Figure 2.4 – Blocks of Contiguous Pixels in Image Partitioning (Crockett, 1997:10)

2.8.1.3. Temporal Parallelism

In this type of parallelism, the fundamental unit of work is a complete image, and each processor is assigned a number of frames to render, along with the data needed to produce theses frames.

2.8.2. Sorting Algorithm

According to Molnar (1994:1-9), a standard, feed-forward rendering pipeline, adapted for parallel rendering is shown on figure 2.5 which consists of two parts. The two principal parts are geometry processing (transformation, clipping, lighting, etc) and rasterization (scan-conversion, shading, and visibility determination). In the figure shown below, processors G perform geometry processing and processors R perform rasterization.

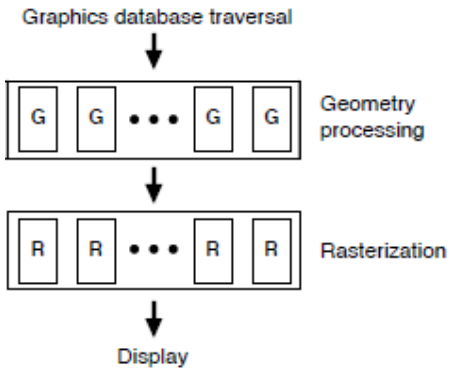

Figure 2.5 – Graphics Pipeline in a Fully Parallel Rendering System (Molnar, 1994:2)

Geometry processing usually is parallelized by assigning each processor a subset of the primitives (objects) in the scene. Rasterization usually is parallelized by assigning each processor a portion of the pixel calculations.

The sort can take place anywhere in the rendering pipeline, during geometry processing (sort-first), between geometry processing and rasterization (sort-middle), or during rasterization (sort-last).

2.8.2.1. Sort-First

The aim is to distribute objects in the early rendering pipeline – during geometry processing – to processor that can do the remaining rendering calculations. The job is done by dividing the screen into disjoint regions and making processors (renderer) which is responsible for all rendering calculations that affect the respective screen regions.

The object then redistributed over an interconnect network to the appropriate renderer and perform the remainder of the geometry-processing and rasterization calculations for the objects.

This redistribution of objects at the beginning of the rendering process is the distinguishing feature of sort-first. However, for some objects, a portion of geometry processing

is done on the wrong renderer. As result, the objects must either be send or recomputed on the new renderer.

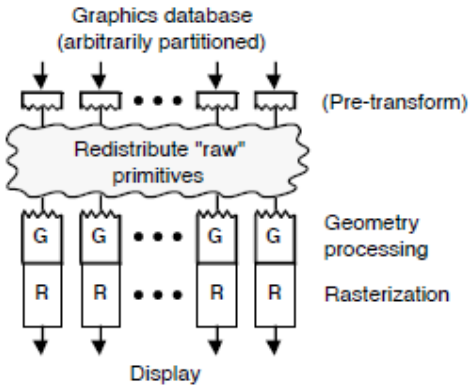

Figure 2.6 – Sort-First (Molnar, 1994:3)

Advantages of sort-first:

- Low communication requirements when the tessellation ratio and the degree of oversampling are high, or when frame-to-frame coherence can be exploited.
- Processors implement entire rendering pipeline for a portion of the screen.

Disadvantages of sort-first:

- Susceptible to load imbalance. Objects may clump into regions, concentrating the work on a few renderers.
- To take advantage of frame-to-frame coherence, retained mode and complex data handling code are necessary.

2.8.2.2. Sort-Middle

The objects are redistributed in the middle of the rendering pipeline – between geometry processing and rasterization. At this point, the objects have been transformed into screen coordinates and are ready for rasterization.

Geometry processors are assigned arbitrary subsets of the objects to be displayed in this system. Rasterizers are assigned a portion of the display screen.

Geometry processors transform, light, etc. their portion of the objects to classify them with respect to screen region

boundaries and then transmit all of the screen-space objects to the right rasterizers during each frame.

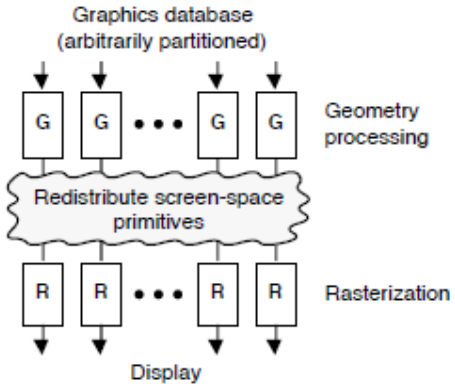

Figure 2.7 – Sort-Middle (Molnar, 1994:3)

Sort-middle system has its advantages:

- General and straightforward; redistribution occurs at a natural place in the pipeline

Sort-middle system disadvantages:

- High communications costs if tessellation ratio is high.
- Susceptible to load imbalance between rasterizers when objects are distributed unevenly over the screen.

2.8.2.3. Sort-Last

This type of sorting defers sorting until the end of the rendering pipeline – after objects have been rasterized into pixels, samples, or pixel fragment. Processors in this system (renderer) are assigned arbitrary subsets of the objects. after that, renderers transmit the pixels over network to compositing processors which resolve the visibility of pixels from each renderer.

Renderers operate independently until the visibility stage, which is the last stage of the rendering pipeline. The network must handle all of the pixel data generated by all of the renderers. This will result in a very high data rates.

Sort-last system can be done in two ways, which are:

a. SL-Sparse

This type of approach minimizes communication by only distributing pixels actually produces by rasterization.

b. SL-Full

This type of approach stores and transfers full image from each renderer.

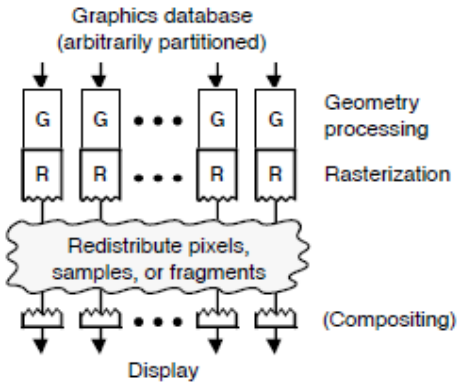

Figure 2.8 – Sort-Last (Molnar, 1994:4)

Sort-last system advantages are:

- Renderers implement the full rendering pipeline and are independent until pixel merging.
- Less prone to load imbalance.
- SL-Full merging can be embedded in a linear network, making it linearly scalable.

Sort-last system disadvantage:

- Pixel traffic may be really high, particularly when oversampling.

2.9. Autodesk 3ds Max

Universitas Bina Nusantara Alam Sutera Visual Communication Design Animation Program students are familiar with Autodesk 3ds Max based on questionnaire (Chapter 3 page 68, Figure 3.12). Therefore, this project is designed to integrate with Autodesk 3ds Max. 3ds Max SDK and MAXScript are used to construct the project's integration unit.

According to Derakhshani (2013:1), Autodesk 3ds Max is a software interface where the user can view and work with the scene.

2.9.1. 3ds Max Rendering

According to Derakhshani (2013:331), rendering is the last step for user in creating a computer generated model, but it is the first step to consider when the user starts to build a scene. During rendering, the computer calculates the scene's surface properties, lighting, shadows, and object movement and then saves a sequence of images.

One included renderer in 3ds Max is Mental Ray Renderer. According to Derakhshani (2013:357), mental ray is a popular renderer that has great capabilities. Mental ray has an ability to generate physically accurate lighting simulations based on indirect principles. Indirect lighting means when light bounces from one object in a scene onto another object. Mental ray's reflections and refractions take on very real qualities when set up and lighted well.

2.9.2. MAXScript

Murdock (2014:1203) stated that MAXScript is a tool that can be used to expand the functionality of 3ds Max. MAXScript can be used to add new features or to customize how 3ds Max behaves, so it can be made to suit the user's custom.

MAXScript is flexible and simple, meaning that it is easy to use and was designed to be an integral part of 3ds Max. Users can order 3ds Max to do something to the scene by typing in a list of instruction that users want 3ds Max to execute.

2.9.3. 3ds Max SDK

Autodesk (2013b) stated that, the 3ds Max SDK Programmer's Guide describes how to use the 3ds Max Software Development Kit to extend the features, functionality, and interface of 3ds Max primarily using C++.

One of the main class references to make plugin is "UtilityObj" class reference. 3ds Max Utility plug-ins are derived from this class.

Methods are provided for editing the utilities parameters and responding to changes in the current selection set. An interface pointer is provided for calling the utility methods provided by MAX.

2.10. Distributed System

According to Kshemkalyani and Singhal (2008:1), a distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved.

The distributed system consists of a set of processors that are connected by a communication network. The communication network provides the facility of information exchange among processors. The communication delay is finite but unpredictable. The processors do not share a common global memory and communicate solely by passing messages over the communication network. There is no physical global clock in the system to which processes have instantaneous access. The communication medium may cause the delivered messages out of order, messages may be lost, garbled, or duplicated due to timeout and retransmission, processors may fail, and communication links may go down. (Kshemkalyani, 2008:39)

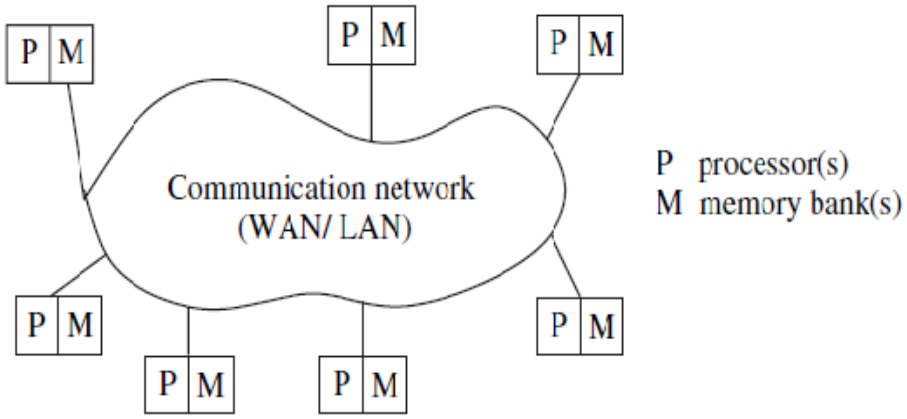

Figure 2.9 - Distributed System Which Connects Processors by a Communication Network (Kshemkalyani, 2008:2)

A distributed system can be characterized as a collection of mostly autonomous processors communicating over a communication network and having the following features:

2.10.1. No Common Physical Clock

This is an important assumption because it introduces the element of “distribution” in the system and gives rise to the inherent asynchrony amongst the processors.

2.10.2. No Shared Memory

This is a key feature that requires message-passing for communication. This feature implies the absence of the common physical clock.

It may be noted that a distributed system may still provide the abstraction of a common address space via the distributed shared memory abstraction. Several aspects of shared memory multiprocessor systems have also been studied in the distributed computing literature.

2.10.3. Geographical Separation

The geographically wider apart that the processors are, the more representative is the system of a distributed system. However, it is not necessary for the processors to be on a wide-area network (WAN). Recently, the network/cluster of workstations (NOW/COW) configuration connecting processors on a LAN is also being increasingly regarded as a small distributed system. This NOW configuration is becoming popular because of the low-cost high-speed off-the-shelf processors now available. The Google search engine is based on the NOW architecture.

2.10.4. Autonomy and Heterogenity

The processors are “loosely coupled” in that they have different speeds and each can be running a different operating system. They are usually not part of a dedicated system, but cooperate with one another by offering services or solving a problem jointly.

2.11. Flow Models of Network

Network could be compared in general, well-known flow models. According to McCabe (2007:180-181), flow models are groups of flows that

exhibit specific, consistent behavior characteristics. Flow models are used to identify and categorize flows in a specific environment. McCabe explain four models of flow in the network, there are:

2.11.1. Peer-to-Peer

The users and applications are fairly consistent in terms of flow behaviors throughout the network. Peer-to-peer model has two important implications, which are:

- Flows in this model cannot be distinguished, which means all of the flows or none of the flows is critical.
- The flows are equivalent and they can be described by a single specification.

Devices communicate directly with each other is considered as peer-to-peer. Peer-to-peer model can be used to describe flows when all users in in a group of network need equal access for an application.

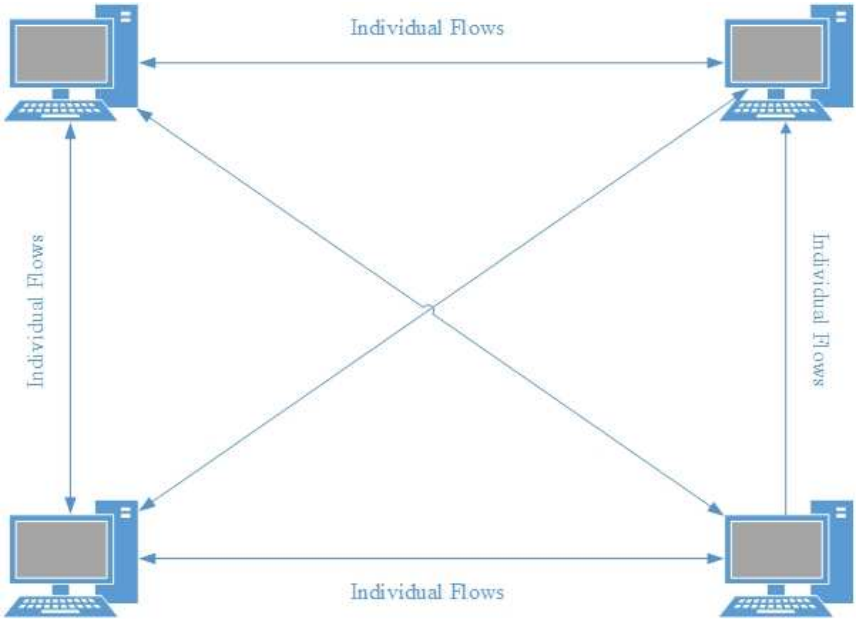

Figure 2.10 – Peer-to-Peer Flow Model (McCabe, 2007:182)

According to Lammie (2012:10), computers connected together in peer-to-peer networks do not have any central, or special, authority. They are all peers, meaning that when it comes to authority, they are all

equals. This means it is up to the computer that has the resource being requested to perform a security check for access rights to its resources.

Peer-to-peer network works well if the number of users of the network is not high, each user could back things up locally, and the network security does not need to be strict.

According to Nagel (2010:1341-1342), every client participating in a peer-to-peer network application must be able to perform the following operations to overcome these problems:

- Able to *discover* other clients.
- Able to *connect* to the other clients.
- Able to *communicate* with other clients.

Systems use the "list on a server" solution by using servers known as *tracker.* Also, in file-sharing systems, any client may act as a server by declaring that it has a file available and registering it with a tracker. In fact, a pure peer-to-peer network does not need any servers at all, just peers.

The connection problem is a more subtle one, and concerns the overall structure of the networks used by a P2P application. If there were a group of clients, all of which can communicate with one another, the topology of the connections between these clients can become extremely complex. Performance could be improved by having more than one group of clients, each of which consists of connections between clients in that group, but not to clients in other groups. If these groups were made locale-based, it will boost the performance, because clients can communicate with each other with fewer hops between networks.

According to Sommerville (2011:499), it is appropriate to use a peer-to-peer architectural model for a system in two conditions:

1. Where the system is computationally intensive and it is possible to separate the processing required into a large number of independent computations.
2. Where the system primarily involves the exchange of information between individual computers on a network and

there is no need for this information to be centrally stored or managed.

According to Kelaskar et al. (2003:1-2), flooding protocols is a method in peer discovery. In flooding protocols, peers flood an overlay network with queries to discover a resource. Strong and widespread reach of discovery characterize flooding protocols. Flooding protocol might give optimal results in a network with a small to average number of peers, but it does not scale well. Furthermore, discovery of peers is not too accurate.

2.11.2. Client-Server

Based on McCabe (2007:184), client-server model is currently the most generally applicable model. The flow in this model is bidirectional between the client and the server in form of requests and responses. The server serves as a data source and the client serves as data sinks.

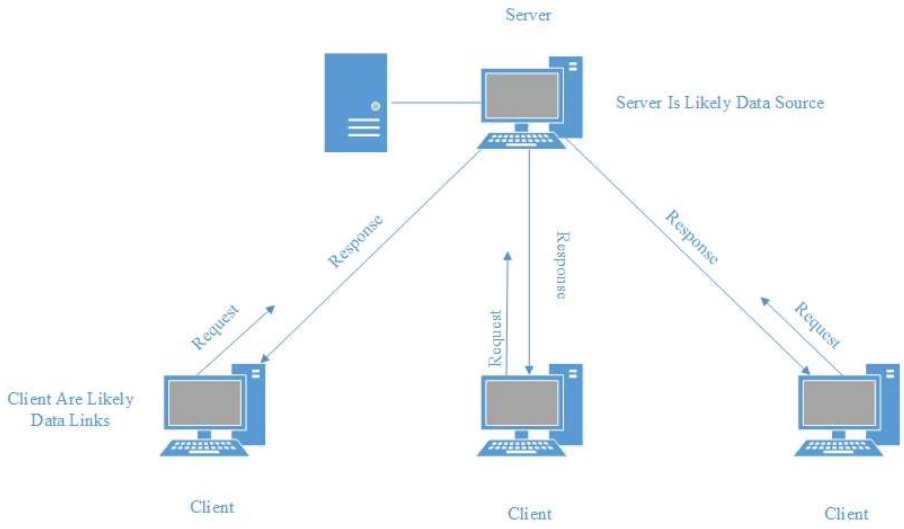

Figure 2.11 – Client-Server Flow Model (McCabe, 2007:184)

2.11.3. Hierarchical Client-Server

According to McCabe (2007:185), hierarchical client-server has the characteristics of a client-server flow model but also has multiple layers, or tiers, between servers. In this model, there can be flows from

a server to another support server or any management devices. The server has a possibility be either data sources and/or data sinks.

The critical flows in this model depend on application behavior. Client-server flows may be the critical flows if the applications are inherently client-server only to support multiple sessions. If the servers communicate each other, then the server-to-server flows may be critical, in addition of the client-server flows critical possibility.

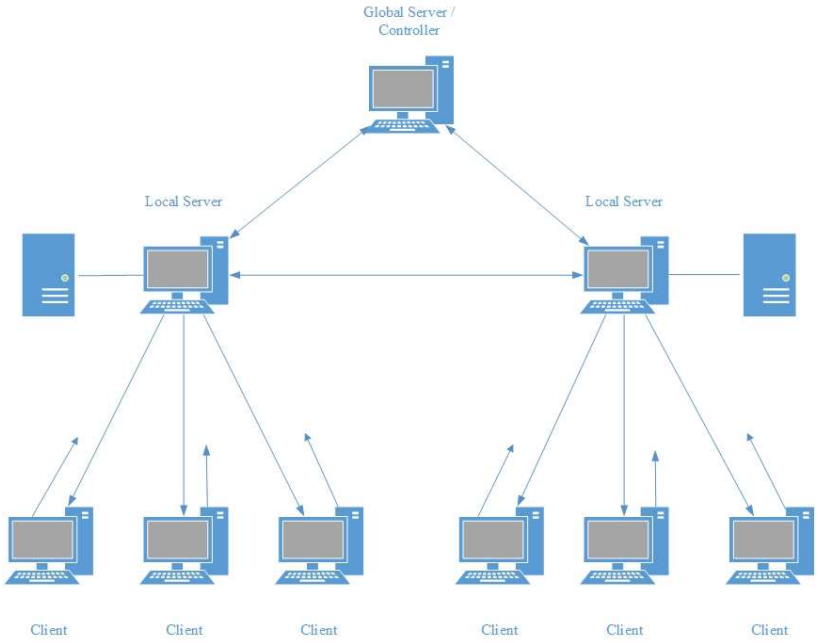

Figure 2.12 – Hierarchical Client-Server Flow Model (McCabe, 2007:186)

2.11.4. Distributed Computing

According to McCabe (2007:188), distributed-computing flow model is the most specialized of the flow models. This flow model can have the inverse of the characteristics of the client-server flow model, or a hybrid of peer-to-peer and client-server flow models. The flows in this model can be client-server but are reversed in direction.

Flow requirements vary between the computing cluster and parallel system model. It depends on the degrees of coupling and the granularity in the task.

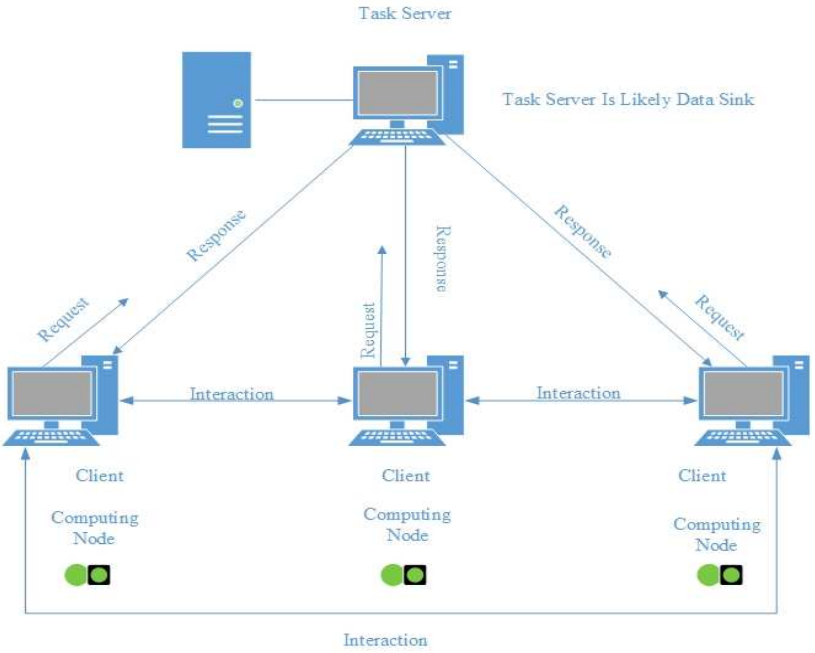

Figure 2.13 – Distributed-Computing Flow Model

2.12. Software Development Life Cycle (SDLC)

According to Dennis (2012:10-15), building an information system is similar to building a house where the owner of the house describes the vision for the house to the developer, then the idea is transformed into sketches and drawings that are shown to the owner and refined until the owner agrees that the pictures depict what he or she wants. Next, a set of detailed blueprints is developed that presents much more specific information about the house. Lastly, the house is build following the blueprints and often with some changes and decisions made by the owner as the house is created.

In order to build an information system using SDLC, a set of four phases are needed, which are Planning Phase, Analysis Phase, Design Phase, and Implementation Phase. Here are some explanations regarding to the phases:

a. Planning Phase

Understanding why an information system should be build is fundamental in planning phase. This phase determine how the project team should do in order to build the information system. There are two steps, which are:

The first step is called project initiation step. It identifies the system's business value to the organization. Basically, it covers about lowering costs and increasing revenues, which comes from the outside of the information system area (e.g. marketing and accounting department) in the form of a system request. The system request itself contains a brief summary of a business need and contains an explanation of how a system that supports the need will create business value. Next, the feasibility analysis is conducted from the person or department who works together with the IS department. There are 3 aspects which are examined by the feasibility analysis:

- Technical feasibility (Can we build it?)
- Economic feasibility (Will it provide business value?)
- Organization feasibility (If we build it, will it be used?)

Project manager directs the project through the whole SDLC by creating a work plan, staffs the project, and puts techniques in place to help the project team control after the project has been approved. The output is a project plan which describes how the project team will go about developing the information system. This step is called project management.

b. Analysis Phase

Analysis phase has the explanation of who will be using the system, what the system will do, where the system will be used and when the system will be used which will be investigated by the project team. There are three steps in analysis phase:

- *Analysis strategy* to guide the project team's efforts. Its goal is to study the current system (*as-is system*) and its problems, and then picturing ways to design a new system (*to-be system*).
- *Requirements Gathering* step is to find a system concept which will be used as a basis to develop a set of business *analysis model* describing how the business will operate when the new system are developed.
- The *system proposal* is created from the analysis, system concept, and models. The *System proposal* is presented to the

project sponsor and other key decision makers whom decide whether the project should continue or not.

The output, the *system proposal* describes what business requirements the new system should meet.

c. Design Phase

The system will be decided of how it should be operated in terms of the hardware, software, and network infrastructure that will be in place in design phase. The UI, forms, and all of the important needs such as files and databases are being decided. There are four steps in design phase:

- Determining the *design strategy* which explains whether the system will be handled by the company's programmers, whether the development will be outsourced to the other firm, or whether the company would buy an existing package.
- Development of the basic *architecture design* that describes the hardware, software, and network infrastructure that will be used in the system.
- Defining what data that will be stored and where it will be stored by developing the database and file specifications.
- Defines the programs that need to be written and what the program will do by developing the *program design.*

The output is the *system specifications* which will be used by the programming team for the next phase, implementation phase. At the end of this phase, the feasibility analysis and project plan are re-examined and revised by the project sponsor and approval committee whether the project will continue or not.

d. Implementation Phase

Implementation phase is the last phase in the SDLC which take the longest and most expensive for most system development. There are three steps:

- The system is built and tested to verify the system performs as it planned. Testing the system would grab more time and attention. This step is called the *construction* step.

- Installing the new system and turn of the old system. There will be a *training plan* to teach the users to use the new system and help manage the changes.
- Establishing a *support plan* for the system which includes a formal or informal post-implementation review, and a systematic way to identify the changes of the systems needed.

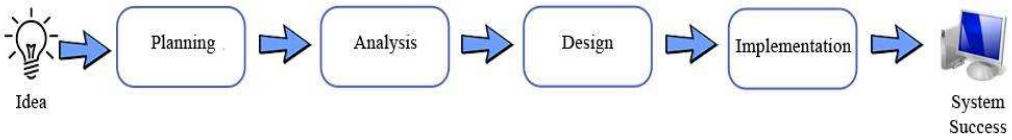

Figure 2.14 - SDLC Phases (Dennis, 2012:11)

2.12.1. Waterfall Model

According to Sommerville (2011, 30-32), the waterfall model is a plan-driven process, which means that the project manager must plan and schedule all of the process activities before starting work on the project. The principal stages of the waterfall are:

a. Requirements Analysis and Definition

The system's services, constraints, and goals are established by consultation with system users. After they are described in detail, they serve as a system specification.

b. System and Software Design

The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. In this phase, it involves in identifying and describing the fundamental software system abstraction and their relationships.

c. Implementation and Unit Testing

The software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

d. 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.

e. Operation and Maintenance

   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.

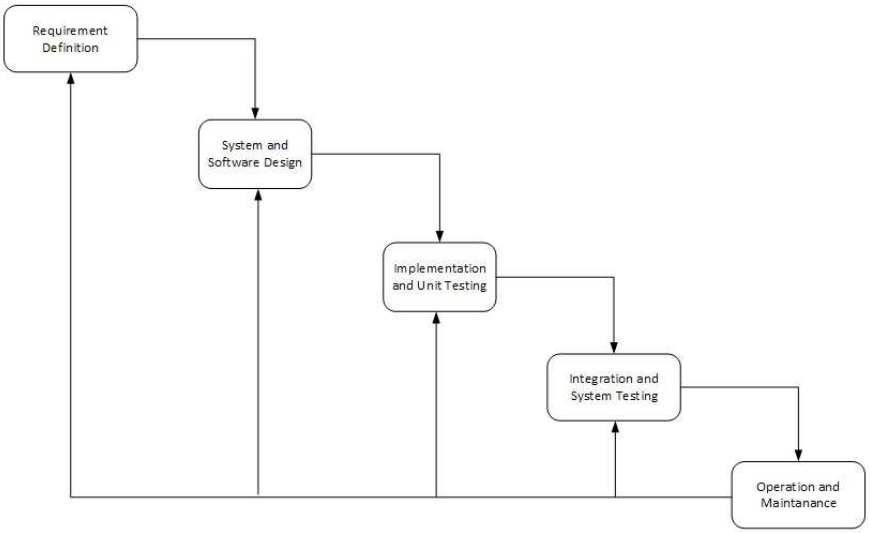

Figure 2.15 – Waterfall Model (Sommerville, 2011:30)

Each phase could not start until the previous phase has been finished. The software process is not a simple linear model but involves feedback from one phase to the other phase. Documentation made in each phase will need to be modified to reflect the changes made.

The cost of producing and approving documents is high and may involve much rework. Because of that, after a small number of iterations, freezing a part of development is normal and the project can move to the next development stages.

The errors in the software requirements were found at the end of the cycle phases. Program and design error emerge and the need for new functionality is found. The project manager must make the changes by repeating previous process stages.

The waterfall model is consistent with other engineering process models and documentation is produced at each phase. Therefore, the process is visible so managers can monitor progress against the development plan. The waterfall should be used when the requirements are well understood and not drastically changing during system development.

2.12.2. Agile Model

Agile model focused in fast development process and deliver output quickly to customers. This model is best used to application development which its system requirements change rapidly during the development process. (Sommerville, 2011:58-59)

Agile methods have been successful for several 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.

Agile model focus on small, tightly integrated teams and will find it hard in scaling them to large systems. Several types of agile methods approach are Extreme Programming and Scrum.

2.12.2.1. Extreme Programming (XP)

Extreme Programming is the best known and most widely used of the agile methods. This approach was developed by pushing recognized good practice, such as iterative development, to ‘extreme’ levels. In XP, new versions of a system may be developed by different

programmers, integrated and tested in the same day. (Sommerville, 2011:64-65)

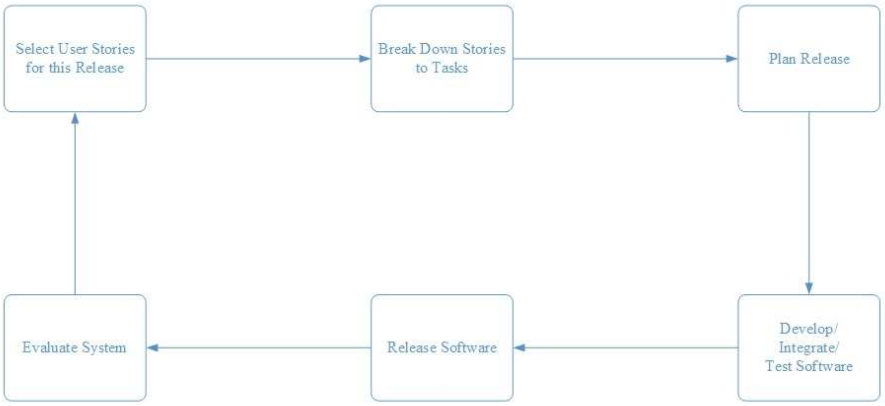

Figure 2.16 – Extreme Programming Release Cycle (Sommerville, 2011:65)

2.12.2.2. Scrum

Scrum is one of development model in agile model. According to Sommerville (2011:72-74), scrum is a general agile method but its focus is on managing iterative development rather than specific technical approaches to agile software engineering.

Scrum has three phases, which are:

a. Outline Planning Phase

   The general objectives for the project and the software architecture design were established.

b. Sprint Cycles

   Each cycle develops an increment of the system in a series of sprint cycles.

c. Project Closure

   Project closure completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project to wrap all projects.

The sprint cycle is the innovative feature of scrum method which is a planning unit where the work to be done is measured, features for development are selected, and software implementation. After this sprint cycle has been done, the completed functionality will be given to stakeholders. Key characteristics of this process are:

1. Sprints are fixed length, normally 2-4 weeks, which correspond to the development of a release of the system in XP.
2. Starting point for planning is the product backlog, which is the list of the work to be done on the project. The customer can ask for new requirements or task 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. The project team moves to the development of the software after the selections are agreed. Daily meetings for all members can be held sometimes to review progress and reprioritize work. During this stage, all communications channeled through the Scrum Master.
5. Each sprint cycle is reviewed and presented to stakeholders then moved to the next sprint cycle.

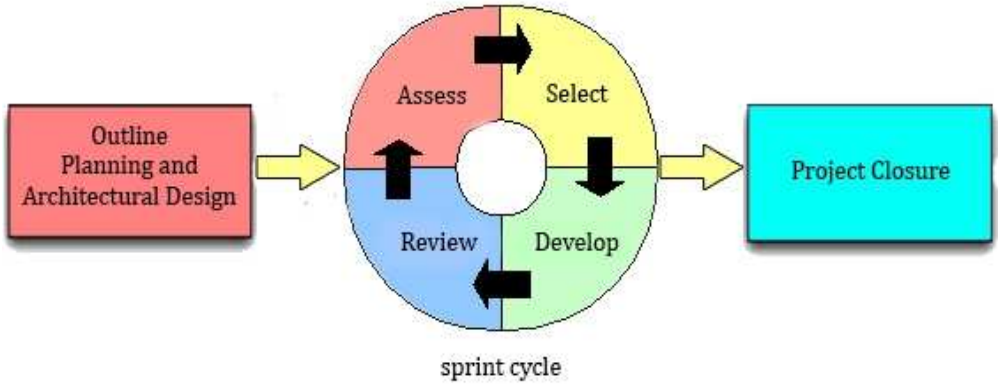

Figure 2.17 – Scrum Process (Sommerville, 2011:73)

2.13. Development Testing

This project uses unit testing for its development testing to test the application is working as it should be.

According to Sommerville (2011:210-221), the development testing include all testing activities that are carried out by the team developing the system. The tester of the software is usually the programmer who developed the software. During its development, testing has three levels of granularity, which are:

2.13.1. Unit Testing

Unit testing is the process of testing program components, such as methods or object classes. Individual functions or methods are the simplest type of component. The tests should be calls to these routines with different input parameters. In testing object classes, the tests need to be designed to provide coverage of all the features of the object, the objective to do it are:

- Test all operations associated with the object.
- Set and check the value of all attributes associated with the object.
- Put the object into all possible states, which means that all events that cause a state change should be simulated

Testing is expensive and time consuming. Therefore, an effective unit test case is important. The effective aspect means:

- The test cases should show that, when used as expected, the component that are being tested does what it is supposed to do.
- If there are defects in the component, these should be revealed by test cases.

2.13.2. Component Testing

Testing composite components should focus on showing that the component interface behaves according to its specification. After it has been verified, the unit tests on the individual objects within the component have been completed.

2.13.3. System Testing

System testing involves integrating components to create a version of the system and then testing the integrated system. In this type of testing, the components are checked through some aspects; compatibility, interaction, and reliable data transfer at the right time.

System testing should focus on testing the interactions between the components and objects that make up a system. This type of testing should discover the component bugs that are only exposed during interaction between components in the system.

System testing can be based on experience of system usage and focus on testing the features of the operational system, for example:

- All system functions that are accessed through menus should be tested.
- Combinations of functions that are accessed through the same menu must be tested.
- All functions must be tested with both correct and incorrect input where user input is provided.

2.14. Flowchart

According to Pressman (2010:299-300), flowchart is like an activity diagram, which is quite simple pictorially. Flowchart has several symbols to be used for building a simple and understandable system's flow. The symbols are a box, a diamond, and arrows. A box is used to indicate a processing step. A diamond represents a logical condition. Arrows show the flow of control. Four structured constructs is illustrated below (Figure 2.18, Figure2.19, Figure 2.20, and Figure 2.21) and each of them has their own usage.

2.14.1. Sequence

Sequence is represented as two processing boxes connected by a line of control.

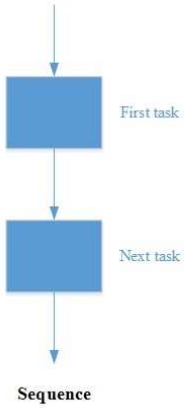

Figure 2.18 – Sequence Constructs (Pressman, 2010:299)

2.14.2. If-then-else

If-then-else is also called as condition, depicted as a decision diamond that, if true, causes then-part processing to occur, and if false, invokes else-part processing.

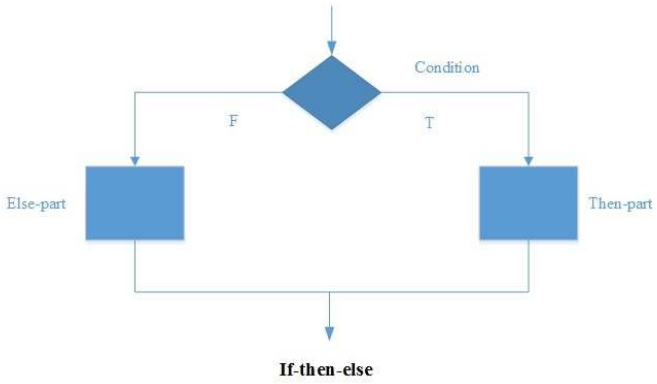

Figure 2.19 – If-then-else Constructs (Pressman, 2010:299)

2.14.3. Selection

The selection is in fact an extension of the if-then-else. A parameter is tested by successive decisions until a true condition occurs and a case part processing path is executed.

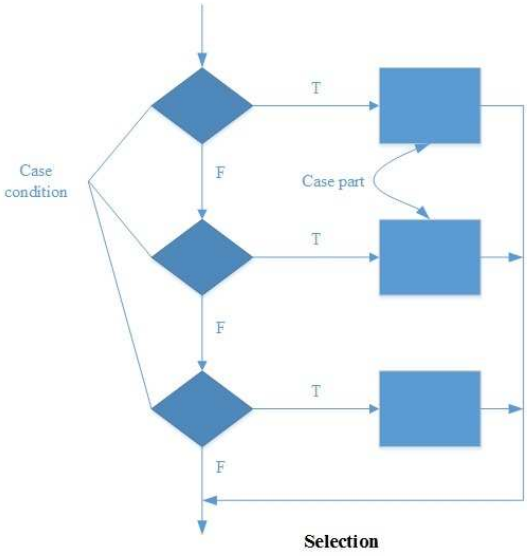

Figure 2.20 – Selection Constructs (Pressman, 2010:299)

2.14.4. Repetition

Repetition is represented using two slightly different forms. The do-while tests a condition and executes a loop task repetitively as long

as the condition is true. A repeat-until executes the loop task first and then tests a condition and repeats the task until the condition fails.

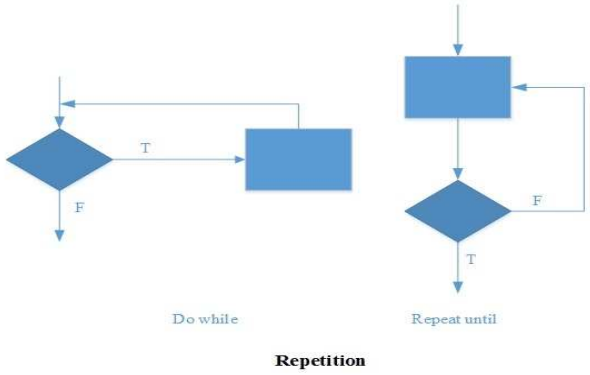

Figure 2.21 – Repetition Constructs (Pressman, 2010:299)

2.15. Unified Modeling Language (UML)

UML purpose is to provide a common vocabulary of object-based terms and diagramming techniques which is sufficient to model any systems development project from analysis to design. (Dennis, 2012:513)

The latest version of UML is version 2.0. There are 14 diagramming techniques for modeling a system, which are:

- Class Diagram

  The relationships between classes modeled in the system are illustrated.
- Object Diagram

  The relationships between objects modeled in the system are illustrated.
- Package Diagram

  UML elements are gathered into one group to form higher level constructs.
- Deployment Diagram

  It mainly shows software components which are being deployed into the physical architecture.

- Component Diagram

  It is used to picture the physical relationships of the software components.
- Composite Structure

  Basically, it defines about the parts relationships in the class.
- Activity Diagram

  It defines the flow of activities in a use case.
- Sequence Diagram

  Picture the object's behavior in a use case with time-based activity.
- Communication Diagram

  Picture the object's behavior in a use case, focuses on communication inside the collaborating objects of an activity.
- Interaction Overview Diagram

  Define the model of the process' flow of control.
- Timing Diagram

  Define the interaction among the objects and the state changes through time.
- Behavioral State Machine Diagram

  Examine class' behavior.
- Protocol State Machine Diagram

  Define the interfaces' dependencies in the class.
- Use Case Diagram

  It illustrates the system and its environment's relationships and also capturing the business requirements.

Not all of these diagrams are necessary to create a whole system, but it depends on what system is being modeled. This project uses this UML diagrams:

2.15.1. Use Case Diagram

Use case diagram represents several "paths" that a user can take while interacting with the system while each path is referred to as a scenario. (Dennis, 2012:517-519)

Use case diagram is the summary of all the use cases gathered in one picture. This diagram is used for understanding the system's functionality. In the early of SDLC, it is usually drawn because it provides simple, brief explanation of how the user and the system would communicate each other.

The components of a use-case diagram are use cases, actor, association relationships, and system boundary.

2.15.1.1. Use Cases

Use case represents the major process that the actor could perform in a system. Use case is oval shaped. Use case has a verb phrase inside that describes its overall usage or function. Use case is placed inside the system boundary. A use case sometimes will use or extend the other use case's functionality inside the diagram which is called includes and extends relationships.

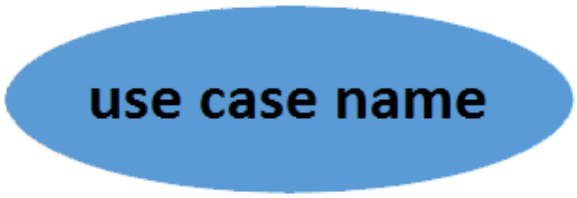

Figure 2.22 - Use Case Example (Source: Dennis, 2012:518)

2.15.1.2. Actor

Actor represents the person or another system that directly interacts with the system and derives value from the system. Actor is symbolized as a stick figure. Actor is placed outside of the system boundary. Actor's role is labeled below the stick figure.

Figure 2.23 - Use Case Actor Example (Source: Dennis, 2012:518)

2.15.1.3. Association Relationship

Association relationship is the connector between actor and use cases. Association relationship indicates which use case does the actor interacts with.

Figure 2.24 - Use Case Association Example (Source: Dennis, 2012:518)

2.15.1.4. System Boundary

System boundary represents internal and external part of the system. System's name is labeled inside of the system boundary's box or at the top of its box.

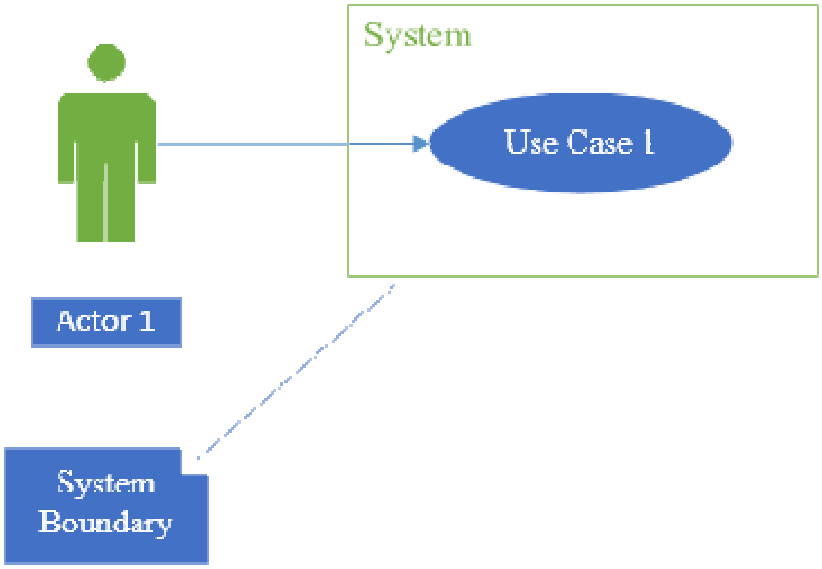

Figure 2.25 - System Boundary Example (Source: Dennis, 2012:518)

2.15.2. Class Diagram

According to Dennis (2012:521-526), class diagram is a static model that supports the static view of the evolving system. In class diagram, classes and the relationship among the classes are shown. Class diagram shows classes with its attributes, behaviors, and states.

The elements of class diagram are class, attribute, method, and association.

2.15.2.1. Classes

Class refers to the people, events, and things about which the system will capture information. Class has three parts. The three parts are the name which is placed at the top, the attribute which is placed at the middle and the methods which is placed at the bottom. Class stores and manages information in the system.

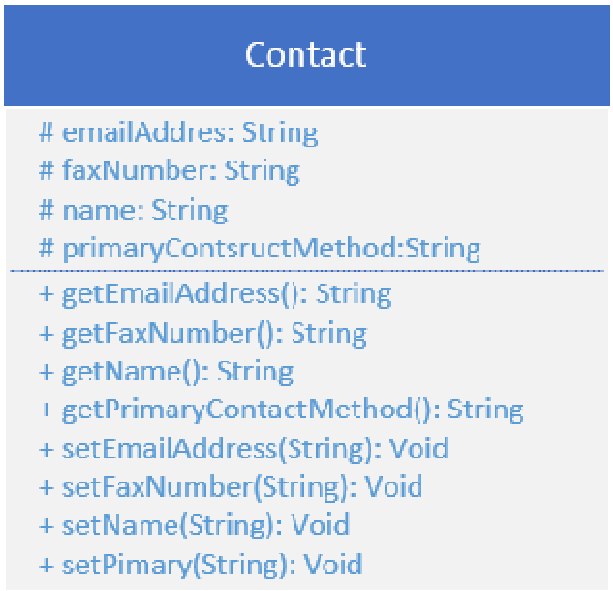

Figure 2.26 - Class Structure (Source: Dennis, 2012:524)

The visibility of an attribute can either be public (+ symbol), protected (# symbol), or private (- symbol). Public attribute is not hidden from any other object. Hidden means other objects can modify its value. A protected attribute is hidden from all other classes except its immediate subclasses. A private attribute is hidden from all other class. The default of visibility is private.

2.15.2.2. Attribute

Attribute is part of the class that describes the information of the object. Attribute can be derived from the other attributes by placing slash notation (/) before attribute's name.

Figure 2.27 - Attribute Example (Source: Dennis, 2012:524)

2.15.2.3. Method

Method represents the actions or functions that a class can perform. Method can be classified as a constructor, query, or an update operation. Method includes parentheses and may contain parameter to perform an operation.

Figure 2.28 - Method Example (Source: Dennis, 2012:524)

2.15.2.4. Association

Association shows the relationships between classes. Association is drawn as a line connecting the classes. Association has multiplicity to show how an instance of an object can be associated with the other instances.

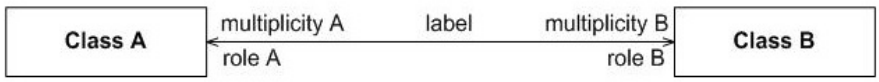

Figure 2.29 - Association with Multiplicity Example (Source: Dennis, 2012:526)

2.15.3. Sequence Diagram

According to Dennis (2012:530-532), sequence diagram illustrates the objects that participate in a use case and the messages that pass between them over time for one use case. Sequence diagram is a dynamic model that supports a dynamic view of the evolving systems. It shows the explicit sequence of messages that are passed between objects in a defined interaction. Sequence diagram is very helpful for

understanding real-time specifications and complex use cases. There are several types of sequence diagram elements, which are:

2.15.3.1. Actor

Actor is a person or system that derives benefit from and is external to the system. Actor usually participates a sequence by sending and/or receiving message. Actor is usually placed across the top of the diagram

Figure 2.30 – Actor (Dennis, 2012:532)

2.15.3.2. Object

Object participates in a sequence by sending and/or receiving message and is placed across the top of the diagram

Figure 2.31 – Object (Dennis, 2012:532)

2.15.3.3. Lifeline

A lifeline denotes the life of an object during a sequence and contains an X at the point at which the class no longer interacts.

Figure 2.32 – Lifeline (Dennis, 2012:532)

2.15.3.4. Focus of Control

Focus of control is a long narrow rectangle placed atop a lifeline and denotes when an object is sending or receiving messages.

Figure 2.33 – Focus of Control (Dennis, 2012:532)

2.15.3.5. Message

Message conveys information from one object to another one.

Figure 2.34 – Message (Dennis, 2012:532)

2.15.3.6. Object Destruction

Object destruction is an X symbol that is placed at the end of an object's lifeline to show that it is going out of existence.

Figure 2.35 – Object Destruction (Dennis, 2012:532)

2.15.4. Activity Diagram

According to Pressman (2010:853-855), activity diagram depicts the dynamic behavior of a system or part of a system through the flow of control between actions that the system performs.

The main component of an activity diagram is an action node, represented by a rounded rectangle, which corresponds to a task performed by the software system. Arrows from one action node to another indicate the flow of control. An arrow between two action nodes means that after the first action is complete the second action begins. A solid black dot surrounded by a black circle is the final node indicating the end of the activity.

A fork represents the separation of activities into two or more concurrent activities. It is drawn as a horizontal black bar with one arrow pointing to it and two or more arrows pointing out from it. Each outgoing arrow represents a flow of control that can be executed concurrently with the flows corresponding to the other outgoing arrows.

A join is a way of synchronizing concurrent flows of control, represented by a horizontal black bar with two or more incoming arrows and one outgoing arrow.

A decision node corresponds to a branch in the flow of control based on a condition. Such a node is displayed as a white triangle with an incoming arrow and two or more outgoing arrows.

Activity diagram can be decorated with swimlanes to indicate how the actions are divided among the participants. Swimlanes are

formed by dividing the diagram into strips or lanes, which corresponds to one of the participants.

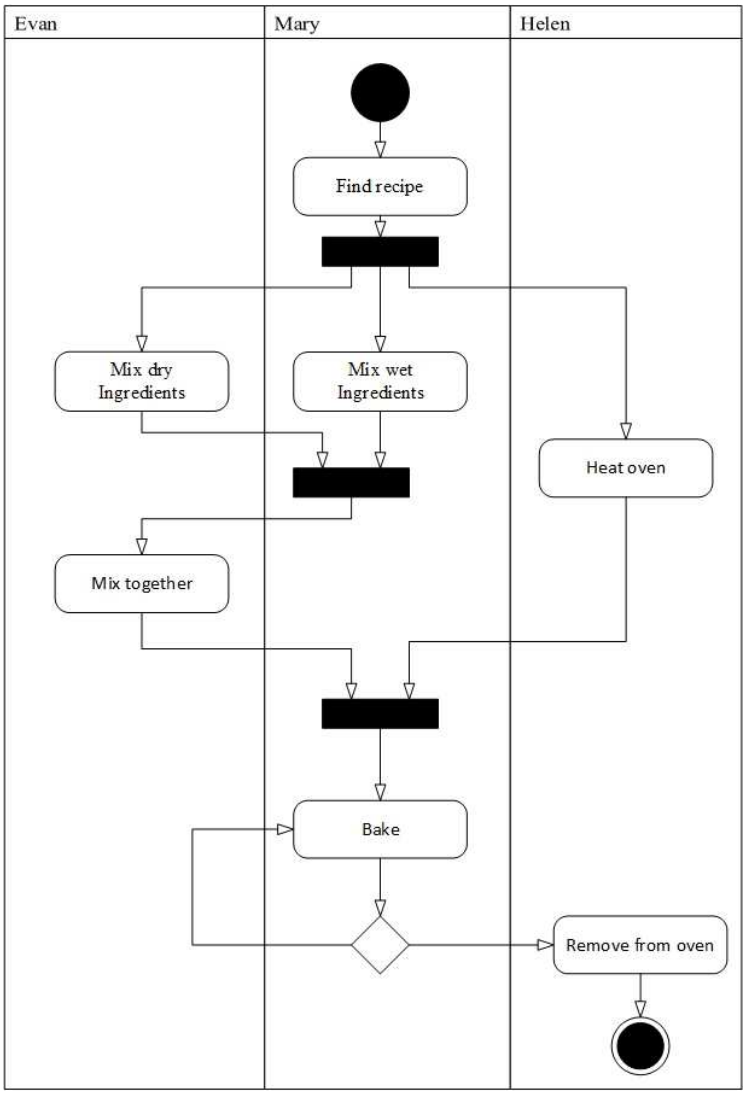

Figure 2.36 – Activity Diagram with Swimlanes (Pressman, 2010:855)

2.16. Visual Studio 2010

According to Nagel (2010:373), Visual Studio 2010 is a fully integrated development environment. It is designed to make the process of writing the code, debugging it, and compiling it to an assembly to be shipped as easy as possible. Visual Studio offers some features, which are:

a. Text Editor

Visual Studio's text editor is quite complex. For example, if the user tries to type the code, it automatically lays out the code by indenting lines, matching start and end brackets of code blocks, and color-coding keywords. It also performs some syntax checking as the user type and underlines the code that causes compilation error.

b. Design View Editor

This editor enables the user to place user-interface and data-access controls in the project.

c. Supporting Windows

Supporting windows allow the user to view and modify aspects of the project, such as the classes in the source code, as well as the available properties for Windows Forms.

d. The Ability to Compile From Within The Environment

The user can call the compile menu inside the Visual Studio environment instead of running it through command line.

e. Integrated Debugger

Integrated debugger allows the user to set the breakpoints and watches on variables from within the environment.

f. Integrated MSDN Help

Visual Studio enables the user to access the MSDN documentation from within the IDE. The user can check a certain compilation error by pressing F1.

g. Access to Other Programs

The user can access a number of other utilities that allows examining and modifying aspects of the computer or network, without having to leave the developer environment.

2.17. .NET Framework

According to Deitel (2011:4), visual programming has become popular because it enables the user to create Windows and web applications easily by using such prepackaged controls as buttons, textboxes, and scrollbars. The .NET Framework executes applications including the .NET Framework Class Library and provides many other programming capabilities.

The details of the .NET Framework are found in Common Language Infrastructure (CLI), which contains information regarding the storage data types. The CLI itself has been standardized, making it easier to implement the .NET Framework for other platforms which made possible for everyone build it by following the specifications.

The Common Language Runtime (CLR) is the central part of the .NET Framework. CLR executes .NET programs and the programs are compiled into machine-specific instructions.

If the .NET Framework is installed on a platform, that platform can run any .NET program. The ability of program to run across multiple platforms is known as platform independence.

2.18. Windows API

According to Hart (2010:xxvii-xxix), the Windows API (Application Programming Interface) is an important factor for application development, frequently replacing the POSIX API as the preferred API for applications targeted at desktop, server, and embedded systems now and for the future.

The windows API is really stable. Programs written in long time can continue to run on the latest Windows releases. The API has expanded even more and there are new features and functions that are useful and sometimes needed. Three examples for Windows API features are:

- The ability to work easily with large files and large 64-bit address spaces.
- Thread pools.
- The new condition variables that efficiently solve an important synchronization problem.

Windows API has two primary mechanisms to do the inter-processing communication which are anonymous pipe and named pipe. Anonymous pipe is character based and half-duplex. Named pipe is better that anonymous pipe. Named pipe is full duplex and message-oriented and allowing networked communication. (Hart, 2010:379)

2.19. State Diagram

According to Sommerville (2011:120-136), state diagram shows how the system reacts to internal and external events. State diagram also show system states and events that cause transitions from one state to another. The flow of data within the system is not shown, but may include additional information on the computations carried out in each state.

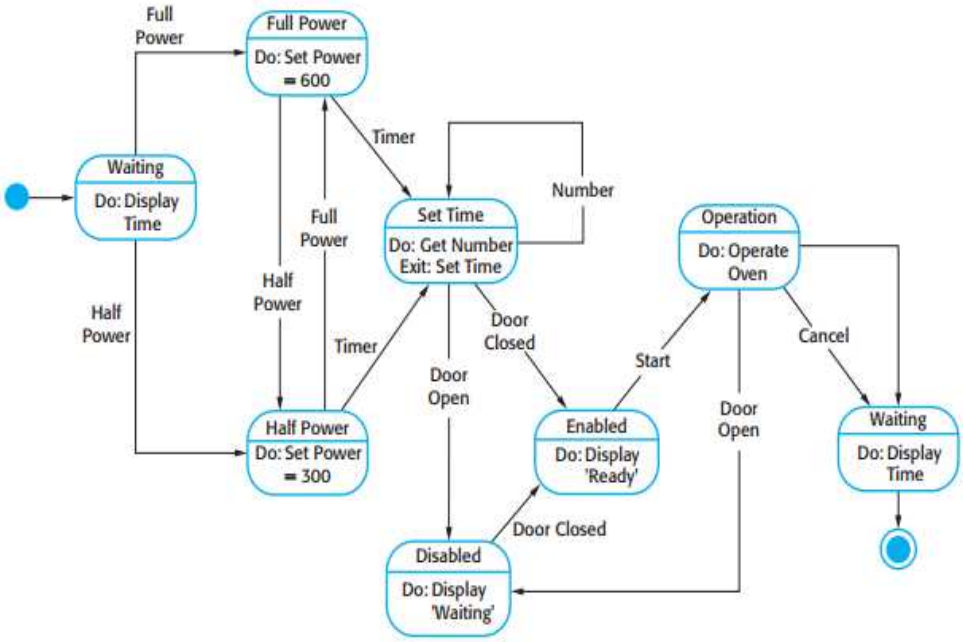

Figure 2.37 – UML State Diagram Notation (Sommerville, 2011:136)