COMPUTER ORGANIZATION AND ARCHITECTURE

Lecture 11

Introduction to System Development and Programming Languages

Dr Victoria Mukami

INTRODUCTION

During this lecture, we review how software systems and applications are developed. We will review the systems development lifecycle and thereafter the program development lifecycle. We will then review the categories of programming language, the evolution of programming languages and the types of languages.

Learning objectives

By the end of this topic, you should be able to:

1. Understand the systems development lifecycle
2. Differentiate between the systems development lifecycle and the program development lifecycle.
3. Highlight the features of each generation of programming languages

OVERVIEW

Throughout this lecture series, we have reviewed the concept of software where we defined software as instructions that tell the computer what to do. With these instructions, users can perform several activities on a computer. These programs whether the operating system or the application software need to be developed in one way or another. The development of these programs is normally the role of designers, analysts, and programmers. An analyst's work is to find out whether there is a need for a system and to do a user analysis of what the system should entail. A designer's work is to pick users requirements and translate them to system requirements that the programmer can work with. The work of a programmer is to code the software and test before implementation. Through this users' we review two terms, the systems development lifecycle, and the program development lifecycle.

SYSTEMS DEVELOPMENT LIFECYCLE

The systems development lifecycle (SDLC) involves five phases through which any system is developed. These phases may vary from organization to organization with some organizations having more than five phases. The five phases of the lifecycle include:

i. Planning
ii. System Analysis
iii. Systems Design

iv. Implementation

v. Support

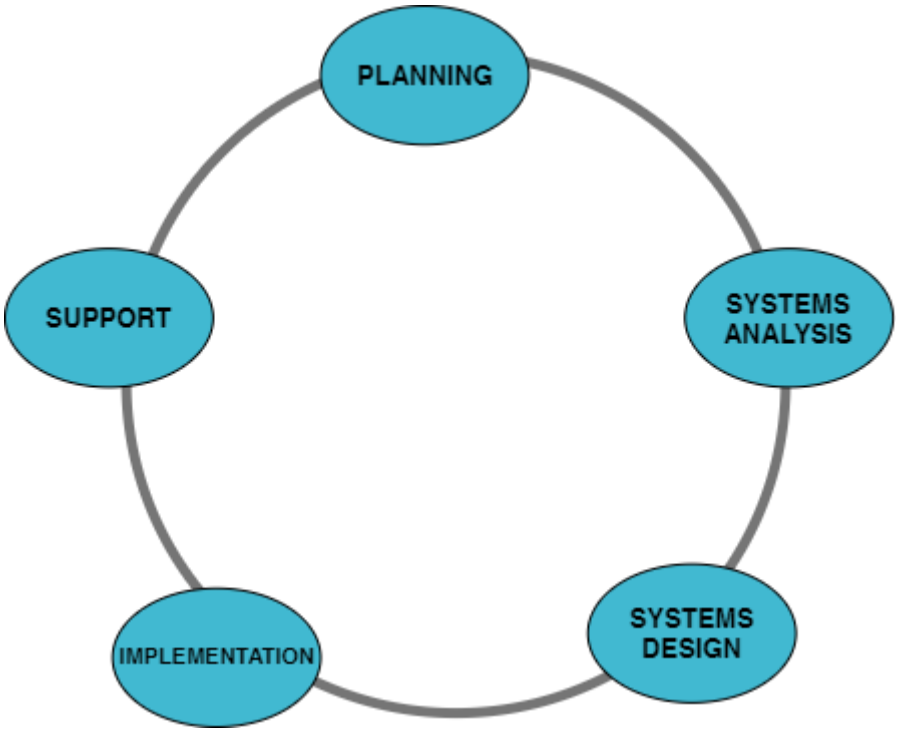

Figure 1: Systems Development Lifecycle

Planning

This is the first phase of any SDLC. It begins with a committee tasked with the needs analysis, where they need to ascertain the need of a system and whether the organization can afford a system and what value the system will bring the organization. This committee also known as the steering committee will receive requests and either approve or reject based on various criteria. This phase also includes documentation that gives approximate costs and benefits of the proposed systems.

System Analysis

During this phase, the proposed system that was accepted during the planning phase is investigated further. This includes a preliminary investigation, a needs analysis and any additional problems that need to be solved. This is conducted by the systems analyst. The system analyst will review any systems currently in place and will do a data collection of the problems with those systems. Data collection may be conducted through an interview, using a questionnaire, observation or doing a document/systems

review. During this data collection, the system analyst will also do a needs analysis where they try to find out what the users might want in the new system. Data collection then leads the way to analysis where all the data collected is analysed to determine any requirements of the new system. Part of the analysis involves coming up with various Unified Modelling Language (UML) diagrams. UML diagrams include diagrams like use case diagrams, context diagrams, data flow diagrams (DFD), class diagrams and entity relationship diagrams (ERD).

System Design

This phase is normally conducted by the designer. The designer's role is to actualize the work of the analyst into a system design. This phase focuses on the visual appeal (how the system will look) and its inner workings. The first thing that is done within the design phase is to determine the technical specifications. This is identifying any software and hardware requirements necessary for the new system. The next is to determine whether the organization will buy or build the new system. If custom off the shelf solutions exists, then it may make sense to buy. On the other hand, if what the organization needs does not exist in off the shelf products, then the organization will likely build or get a vendor to build the system. The organization will then get the proposals of various vendors and review them to determine which vendor to go along with.

Part of the design phase includes coming up with specifications for the new system. This will involve the various modules that will be built for the new system, database specifications and the various inputs and outputs of the system. This could be presented in various designs or screens or through a prototype to show a draft version of the system.

Implementation

This phase is dependent on what the previous outcome was. Whether the organization decided to build or buy. This is whether the vendor will develop or deliver the off the shelf system then implement it within the organization. This is only possible after any new hardware necessary for the new system has been purchased. Some of the tasks of the implementation phase include data migration (moving data from any old system to the new one), testing the inner workings of the system, documentation of the

system, training of the users and finally converting to the new system. This conversion is done using various methods.

Direct cutover method: the users stop using the old system and start using the new system. One of the benefits of using this method is the lower costs of moving from one system to the next. The drawback of this method is that it is extremely risky [1] and can disable the entire organization if testing of the system was not thorough.

Parallel method: here the old and new systems run in parallel until a time when the organizations feel that they can move entirely to the new system. A benefit of this is that any problems within the system can be fixed without disabling the entire organization. Of course, a drawback is higher transition costs of moving from the old to the new system.

Phased conversion: in this method, the new system is introduced in phases throughout the organization. This is an introduction of various modules of the system, one at a time and could either use a phased or parallel approach.

Pilot approach: This is where the organization did not have a system, to begin with, and are implementing the new system. With a pilot approach, the system is introduced at one location or one department and after a period it is introduced to other locations. This gives time for the pilot to be tested in one location.

Support

This is also known as the maintenance phase. This is viewed as an ongoing process [4] where once the system is implemented, it continues until the point where a new system is proposed (the end of the current system). Support also involves monitoring the system performance and ensuring the security of the system is maintained.

PROGRAM DEVELOPMENT LIFECYCLE

We have reviewed in the previous section, the systems development life cycle which has five phases. The program development life cycle is an expansion of the implementation phase of the SDLC. The program development lifecycle (PDLC) has 5 phases.

i. Requirements gathering
ii. Design

iii. Coding
iv. Implementation
v. Testing

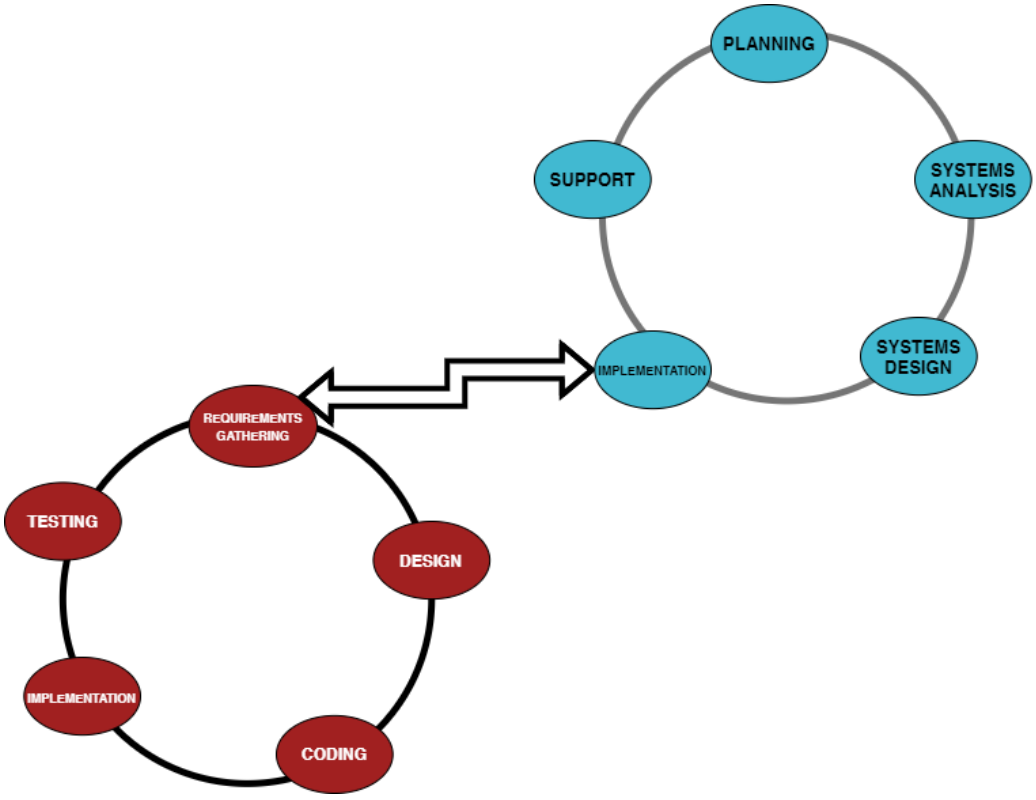

Figure 2: Program Development Lifecycle as a part of the SDLC [1]

These phases are like the SDLC, where the requirement's gathering phase is like the analysis phase. The design, coding, implementation, and testing phases are alike to the phases within the SDLC with testing being isolated from the implementation. These five steps form a loop until a final system is implemented companywide [1].

LANGUAGES OF APPLICATION DEVELOPMENT

A programming language is defined as a set of rules, words, symbols, and codes used to write computer programs [4]. Many different types of programming languages exist based on the way they are used to develop applications. Programming languages are developed based on their need or the problem they intend to solve. Two main classifications of programming languages exist.

Low-level languages

These are languages that were developed and designed to run on the processor directly. These languages are also called machine languages. Programmers would

write programs using the base code that computers understood i.e. 1s and 0s. Low-level languages are dependent on a processor and are only developed for the specific processor. Two main types of languages exist as low-level.

Machine language: These programs are developed using 0s and 1s. These programs work with the processor directly and do not require any translation for them to be understood by the processor. These languages are no longer used, however, modern-day programming languages have to be converted into machine languages for the processor to work on it. An example of machine language

```
00000000      Stop Program
```

Assembly language: This language replaced machine language with names and symbols that were easier to remember by developers [4]. An example of assembly language code includes:

```
mov #COUNT,r5      start the counter
```

High-level languages

These languages are problem-oriented. They do not depend on the processor to run, instead, they rely on the problem that needs to be solved. High-level languages are easy to understand and can run on different platforms. These languages are less memory efficient as compared to low-level languages. Several high-level languages exist including C++, C#, Java, and Python. High-level languages used today need to be translated into a form called machine language. This allows the processor to understand the instructions and codes. There are two ways in which this is done.

Interpret the program: this uses an interpreter to translate the instructions in various languages. An interpreter will read the source code, a line at a time and translate it into a form understandable by the processor.

Compile the program: this means transforming the program written by a programming language from source code to object code. This is normally done by a compiler. The compiler will transform the entire code into machine code.

GENERATIONS OF PROGRAMMING LANGUAGES

There are five generations of programming languages.

First-generation language: this is a low-level language and are also known as 1GL. These languages were machine-dependent. Machine language is the first-generation language.

Second-generation language: this is a low-level language and are also known as 2GL. These languages were aimed at ensuring programmers could remember various codes. Assembly language is a second-generation language.

Third-generation language: also known as 3GL and are high-level languages. These languages were developed to overcome the challenges of 1GL and 2GL languages. Examples of languages included: Fortran, COBOL, C++, and C.

Fourth-generation language: also known as 4GL and were developed to reduce time, cost, and effort. Most fourth-generation languages were used to create and access databases and for scripting purposes. Examples include SQL and Ruby.

Fifth-generation language: also known as 5GL and these languages have visual tools to develop a program. An example of a 5GL is Visual Basic.

PROGRAMMING LANGUAGE SYNTAX

The following section shows how various programming language syntax looks like when a user wants to display the words – Hello World.

Assembly Language

```
section         .text
   global _start     ;must be declared for linker (ld)
_start:             ;tells linker entry point
   mov edx,len     ;message length
   mov ecx,msg     ;message to write
   mov ebx,1       ;file descriptor (stdout)
   mov eax,4       ;system call number (sys_write)
   int    0x80        ;call kernel
   mov eax,1       ;system call number (sys_exit)
   int    0x80        ;call kernel
section         .data
msg db 'Hello, world!', 0xa  ;string to be printed
```

```
len equ $ - msg      ;length of the string
```

C

```
#include <stdio.h>
int main(void)
{
printf("hello, world\n");
return 0;
}
```

C++

```
#include <iostream>
int main()
{
std::cout << "Hello World!" << std::endl;
return 0;
}
```

Java

```
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Python

```
print("Hello, World!")
```

SUMMARY

Throughout this lecture, we have done a review of the systems development life cycle. We then reviewed the program development life cycle and its relation to the systems development life cycle. Next, we reviewed the classifications of programming languages and specifically the low-level and high-level languages. We also learnt about the different generations of programming languages. Finally, we looked at various types of syntax from different programming languages.

DISCUSSION TOPIC

Programming languages have gone through the same transformation as computers and even have five generations. The fifth generation of computers is based on Artificial Intelligence. This also means that computer languages need to get where they can attempt to allow programmers to program without knowing the various rules. Do web research and unearth how programming languages have adopted AI and are mimicking human language.

REFERENCES

[1] G. Shelly and M. Vermaat, Discovering Computers — Fundamentals: Your Interactive Guide to the Digital World. Boston, MA: Course Technology, 2012.

[2] W. Stallings, Computer Organization and Architecture Designing for Performance. Hoboken, NJ: Pearson Education, Inc, 2016.

[3] A. Evans, K. Martin and A. Poatsy, Technology in Action. New York, NY: Pearson, 2020

[4] D. Morley and C. Parker, Understanding Computers: Today and Tomorrow. Boston, MA: Course Technology, 2017.