DEPARTMENT OF COMPUTER SCIENCE AND MATHEMTICS
COLLEGE OF BASIC AND APPLIED SCIENCES
COURSE OUTLINE

SEN 201 Introduction to Software Engineering (2 Units: LH 30)

Semester: Second Semester 2024/2025 Session, Oct – Feb, 2025

Lecturer: Olumoye Mosud, PhD (Associate Professor)

Office: Department of Computer Science,
College of Basic and Applied Sciences (CBAS),
E:mail: yomosud@mtu.edu.ng & myolumoye@yahoo.com

SEN 201: Software Engineering (2 Units: LH 30)

Learning Outcomes

At the end of this course, students should be able to:

1. describe the concept of the software life cycle;
2. explain the phases of requirements analysis, design, development, testing and maintenance in a typical software life cycle;
3. differentiate amongst the various software development models;
4. utilize UML for object-oriented analysis and design;
5. describe different design architectures;
6. explain the various tasks involved in software project management; and
7. describe the basic legal issues related to Software Engineering.

Course Contents

Software Engineering concepts and principles. Design, development and testing of software systems. Software processes: software lifecycle and process models. Process assessment models. Software process metrics. Life cycle of software system. Software requirements and specifications. Software design. Software architecture. Software metrics. Software quality and testing4. Software architecture. Software validation. Software evolution: software maintenance; characteristics of maintainable software; re-engineering; legacy systems; software reuse. Software Engineering and its place as a computing discipline. Software project management: team management; project scheduling; software measurement and estimation techniques; risk analysis; software quality assurance; software configuration management. Software Engineering and law.

CHAPTER ONE
BASIC CONCEPT OF SOFTWARE ENGINEERING

1.1 Concept of Software Engineering

Software engineering is a discipline that allows us to apply engineering and computer science concepts in the development and maintenance of reliable, usable, and dependable software. The concept of software engineering was first discussed at the 1968 NATO Science Committee in Germany. There are several areas to focus on within software engineering, such as design, development, testing, maintenance, and management. Software development outside of the classroom is a very complex process, mostly because real-world software is much larger and more complex.

1.2 Software Engineering Methodology

The body of methods, rules, postulates, procedures, and processes that are used to manage a software project are collectively referred to as a software engineering methodology.

1.2.1 Definitions of Software Engineering.

Engineering is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches. In other words, it is the application of engineering to software.

Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

1.2.2 Sub-disciplines of Software Engineering

Software engineering can be divided into ten sub-disciplines. They are as follows:

- **Software requirements:** The elicitation, analysis, specification, and validation of requirements for software.

- **Software design:** Software Design consists of the steps a programmer should do before they start coding the program in a specific language. It is usually done with Computer-Aided Software Engineering (CASE) tools and use standards for the format, such as the Unified Modeling Language (UML).

- **Software development:** It is construction of software through the use of programming languages.

- **Software testing Software Testing** is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test.

- **Software maintenance:** This deals with enhancements of Software systems to solve the problems they may have after being used for a long time after they are first completed.

- **Software configuration management:** is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines.

- **Software engineering management:** The management of software systems borrows heavily from project management.

- **Software development process:** A **software development process** is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

- **Software engineering tools**, (CASE which stands for Computer Aided Software Engineering) CASE tools are a class of software that automates many of the activities involved in various life cycle phases. CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.

- **Software quality** The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

1.3 Software Engineering Goals and Principles

1.3.1 Goals

Software Engineering as the application of engineering to software has overall goal to easily support changes to software requirements over the system's life. Stated requirements when they are initially specified for systems are usually incomplete. Apart from accomplishing these stated requirements, a good software system must be able to easily support changes to these requirements over the system's life. Therefore, a major goal of software engineering is to be able to deal with the effects of these changes.

The software engineering goals include:

- **Maintainability:** Changes to software without increasing the complexity of the original system design should be possible.

- **Reliability: The** software should be able to prevent failure in design and construction as well as recover from failure in operation. In other words, the software should perform its intended function with the required precision at all times.

- **Efficiency:** The software system should use the resources that are available in an optimal manner.

- **Understand ability*:*** The software should accurately model the view the reader has of the real world. Since code in a large, long-lived software system is usually read more times than it is written, it should be easy to read at the expense of being easy to write, and not the other way around.

1.3.2 Principles

Sounds engineering principles must be applied throughout development, from the design phase to final fielding of the system in order to attain a software system that satisfies the above goals. These include:

- **Abstraction:** The purpose of abstraction is to bring out essential properties while omitting inessential detail. The software should be organized as a ladder of abstraction in which each level of abstraction is built from lower levels. The code is sufficiently conceptual so the user need not have a great deal of technical background in the subject. The reader should be able to easily follow the logical path of each of the various modules. The decomposition of the code should be clear.

- **Information Hiding:** The code should include no needless detail. Elements that do not affect other segment of the system are inaccessible to the user, so that only the intended operations can be performed. There are no "undocumented features".

- **Modularity:** The code is purposefully structured. Components of a given module are logically or functionally dependent.

- **Localization:** The breakdown and decomposition of the code is rational. Logically related computational units are collected together in modules.

- **Uniformit*y:*** The notation and use of comments, specific keywords and formatting is consistent and free from unnecessary differences in other parts of the code.

- **Completeness:** Nothing is deliberately missing from any module. All-important or relevant components are present both in the modules and in the overall system as appropriate.

- **Confirm ability*:*** The modules of the program can be tested individually with adequate rigor. This gives rise to a more readily alterable system, and enables the reusability of tested components.

1.4 Software Engineering vs. Computer Science

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software. Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

1.5 Attributes of a Good Software

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

a. Maintainability: Software must evolve to meet changing needs
b. Dependability: Software must be trustworthy
c. Efficiency: Software should not make wasteful use of system resources
d. Usability: Software must be usable by the users for which it was designed
e. Robustness: Software should fail only under extreme conditions
f. Portability: Should be possible to move from one environment to another

1.6 Software Process

A software process is a set of activities and associated results whose goal is the development or evolution of software product Generic activities in all software processes are:
Specification - what the system should do and its development constraints
Development - production of the software system
Validation - checking that the software is what the customer wants
Evolution - changing the software in response to changing demands

CHAPTER TWO
SOFTWARE DEVELOPMENT AND TESTING

2.1 Concept of Software Development?

Software development provides a series of steps for programmers to create computer programs. This process makes up the phases in the software development life cycle. Understanding the software development method offers vast opportunities in the IT industry.

2.1.1 What is software development?

Software development refers to a set of computer science activities that are dedicated to the process of creating, designing, deploying, and supporting software. Software development is the process programmers use to build computer programs. The process, also known as the Software Development Life Cycle (SDLC), includes several phases that provide a method for building products that meet technical specifications and user requirements. The SDLC provides an international standard that software companies can use to build and improve their computer programs. It offers a defined structure for development teams to follow in the design, creation and maintenance of high-quality software. The aim of the IT software development process is to build effective products within a defined budget and timeline.

2.1.2 Key Steps In The Software Development Process

There are six major steps in the software development life cycle, including:

i. Needs identification

Needs identification is a market research and brainstorming stage of the process. Before a firm builds software, it needs to perform extensive market research to determine the product's viability. Developers must identify the functions and services the software should provide so that its target consumers get the most out of it and find it necessary and useful. There are several ways to get this information, including feedback from potential and existing customers and surveys. The IT teams and other divisions in the company must also discuss the strengths, weaknesses and opportunities of the product. Software development processes start only if the product satisfies every parameter necessarily for its success.

ii. Requirement analysis

Requirement analysis is the second phase in the software development life cycle. Here, stakeholders agree on the technical and user requirements and specifications of the proposed product to achieve its goals. This phase provides a detailed outline of every component, the scope, the tasks of developers and testing parameters to deliver a quality product. The requirement analysis stage involves developers, users, testers, project managers and quality assurance. This is also the stage where programmers choose the software development approach such as the waterfall or V model. The team records the outcome of this stage in a Software Requirement Specification document which teams can always consult during the project implementation.

iii. Design

Design is the third stage of the software development process. Here, architects and developers draw up advanced technical specifications they need to create the software to requirements. Stakeholders will discuss factors such as risk levels, team composition, applicable technologies, time, budget, project limitations, method and architectural design.The Design Specification Document (DSD) specifies the architectural design, components, communication, front-end representation and user flows of the product. This step provides a template for developers and testers and reduces the chances of flaws and delays in the finished product.

iv. Development and implementation

The next stage is the development and implementation of the design parameters. Developers code based on the product specifications and requirements agreed upon in the previous stages. Following company procedures and guidelines, front-end developers build interfaces and back-ends while database administrators create relevant data in the database. The programmers also test and review each other's code. Once the coding is complete, developers deploy the product to an environment in the implementation stage. This allows them to test a pilot version of the program to make performance match the requirements.

v. Testing

The testing phase checks the software for bugs and verifies its performance before delivery to users. In this stage, expert testers verify the product's functions to make sure it performs according to the requirements analysis document. Testers use exploratory testing if they have experience with that software or a test script to validate the performance of individual components of the software. They notify developers of defects in the code. If developers confirm the flaws are valid, they improve the program, and the testers repeat the process until the software is free of bugs and behaves according to requirements.

vi. Deployment and maintenance

Once the software is defect-free, the developers can deliver it to customers. After the release of a software's production version, the IT software development company creates a maintenance team to manage issues clients encounter while using the product. Maintenance can be a hot-fix if it is a minor issue but severe software failures require an update.

2.1.3 Types of Software

Software belongs to three main groups based on their use and application. Here are the popular categories of software.

i. System software

Also called operating system or OS, system software is the program your computer uses to translate input commands into machine-readable language. The operating system controls a computer's hardware components. Examples of popular operating systems used in personal computers include the Windows OS from Microsoft, Mac OS used in Apple MacBook and the

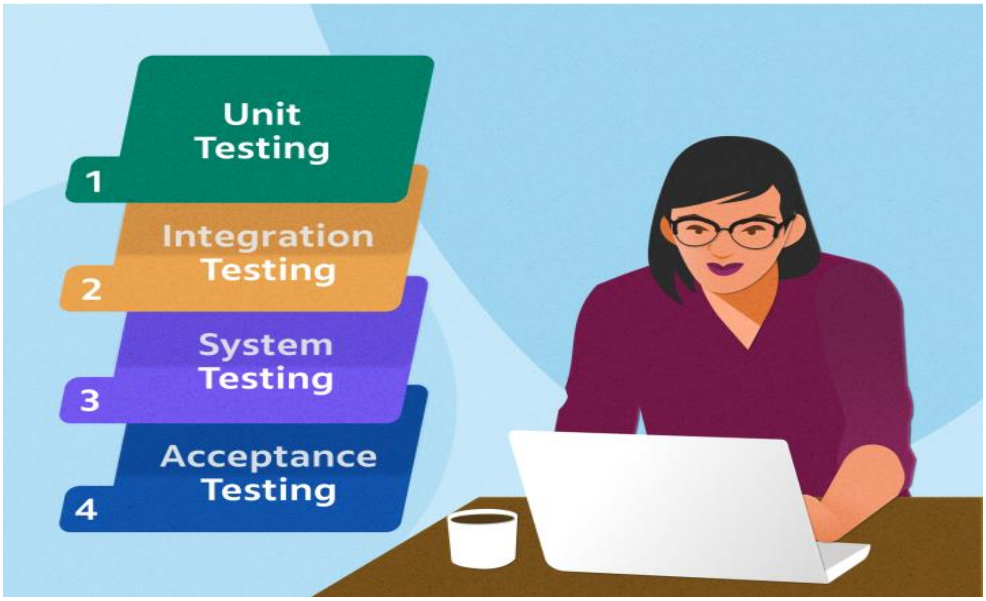
Linux-based Ubuntu. Web servers use the Apache OS while the UNIX operating system is used to build proprietary systems.

ii. Application software
This is the application most people use to perform tasks on their computers and smartphones. Popular examples include word processing apps, internet browsers, media players, photo editing tools, anti-virus and even software-as-service (SAS) products.

iii. Programming languages
This is the programming language used to create software. It is used only by coders to create programs. Programming languages include Java, C++, PHP and Simlab.

2.2 Software testing
Software testing is the process of assessing the functionality of a software program. The process checks for errors and gaps and whether the outcome of the application matches desired expectations before the software is installed and goes live.

Testing provides the answer to the problem "*will the system produce the desired result under known conditions*"? Comprehensive testing must be carried out on the designed system to ascertain whether it produces the right results. Testing is a time-consuming exercise, and costly because as much as fifty percent of the entire development budget may be spent at this stage. Testing has to be carefully prepared, result reviewed and correction made where necessary. In some cases, part of the system may have to be redesigned.



Testing an information system can be grouped into the following activities.

- ***Unit Testing***: This is also referred to as program testing. It is the process of testing, each unit or program that makes up a system separately. The purpose of this testing is to guarantee that programs are error free, but this may be practically difficult to achieve.

- **Integration testing** -- also known as integration and testing (I&T) -- is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity. Or **Integration testing.** This groups together two or more modules of an application to ensure they function collectively. This type of testing also reveals interface, communication and data flow defects between modules.

- ***System Testing***: It involves testing the functioning information system as a whole whether the units or modules will function together as planned. Some examined areas include performance time, capacity for file storage, recovery and restart capabilities.

- ***Acceptance Testing***: This test provides final clearance or certification that the system can commence operation in a production setting. Systems tests are evaluated by end-users and reviewed by the management. When all parties are satisfied with the new system that it conforms to their specifications and standards, it is then formally accepted for installation.

2.2.1 Why is software testing important?

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully, it will remove all the errors from the software. Software testing is the culmination of application development through which software testers evaluate code by questioning it. This evaluation can be brief or proceed until all stakeholders are satisfied. Software testing identifies bugs and issues in the development process so they're fixed prior to product launch. This approach ensures that only quality products are distributed to consumers, which in turn elevates customer satisfaction and trust.

To understand the importance of software testing, consider the example of Starbucks. In 2015, the company lost millions of dollars in sales when its point-of-sale (POS) platform shut down due to a faulty system refresh caused by a software glitch. This could have been avoided if the POS software had been tested thoroughly. Nissan also suffered a similar fate in 2016 when it recalled more than 3 million cars due to a software issue in airbag sensor detectors.

The following are important reasons why software testing techniques should be incorporated into application development:

- **Identifies defects early.** Developing complex applications can leave room for errors. Software testing is imperative, as it identifies any issues and defects with the written code so they can be fixed before the software product is delivered.
- **Improves product quality.** When it comes to customer appeal, delivering a quality product is an important metric to consider. An exceptional product can only be delivered if it's tested effectively before launch. Software testing helps the product pass quality assurance (QA) and meet the criteria and specifications defined by the users.
- **Increases customer trust and satisfaction.** Testing a product throughout its development lifecycle builds customer trust and satisfaction, as it provides visibility into the product's strong and weak points. By the time customers receive the product, it has been tried and tested multiple times and delivers on quality.
- **Detects security vulnerabilities.** Insecure application code can leave vulnerabilities that attackers can exploit. Since most applications are online today, they can be a leading vector for cyber attacks and should be tested thoroughly during various stages of application development. For example, a web application published without proper software testing can easily fall victim to a cross-site scripting attack where the attackers try to inject malicious code into the user's web browser by gaining access through the vulnerable web application. The nontested application thus becomes the vehicle for delivering the malicious code, which could have been prevented with proper software testing.
- **Helps with scalability.** A type of nonfunctional software testing process, scalability testing is done to gauge how well an application scales with increasing workloads, such as user traffic, data volume and transaction counts. It can also identify the point where an application might stop functioning and the reasons behind it, which may include meeting or exceeding a certain threshold, such as the total number of concurrent app users.
- **Saves money.** Software development issues that go unnoticed due to a lack of software testing can haunt organizations later with a bigger price tag. After the application launches, it can be more difficult to trace and resolve the issues, as software patching is generally more expensive than testing during the development stages.

2.2.2 Best practices for software testing

There's more to software testing than running multiple tests. It also entails using a specific strategy and a streamlined process that helps to carry out these tests methodically. To improve the performance and functionality of any application or product, software best practices should always be followed.

The following are a few best practices to consider to help ensure successful software testing projects:

- **Incorporate security-focused testing.** Security threats are constantly evolving. To protect software products from digital threats, security-focused tests should be conducted along with regular software tests. Penetration testing or ethical hacking can help organizations evaluate software integrity from a security standpoint and understand any weaknesses.
- **Involve users.** Since users are the best judge of a software product, developers need to keep the communication channels open with them. Asking open-ended questions -- such as what issues users run into while using the product and the type of features they would prefer to see -- can help conduct testing from the user's perspective. Creating test accounts in production systems that simulate the user experience is also a great way to incorporate their feedback for successful software testing.
- **Keep the future in mind.** The world of technology is constantly evolving and any new product on the market should be scalable and adaptable to changing demands. Before developing a product, developers should keep the future and adaptability features in mind. This can be ensured by the type of architecture used and the way the software is coded. A futuristic product shouldn't only be tested for bugs and vulnerabilities, but for scalability factors as well.
- **Programmers should avoid writing tests.** Tests are typically written before the start of the coding phase. It's best practice for programmers to avoid writing those tests, as they can be biased toward their code or might miss other creative details in the test sets.
- **Perform thorough reporting.** Bug reporting should be as detailed as possible so people responsible for fixing the issues can decipher them easily. A successful report should be balanced and reflect on the severity of issues, prioritize the fixes and include suggestions to prevent those bugs from reappearing.
- **Divide tests into smaller fractions.** Smaller tests save time and resources especially in environments where frequent testing is conducted. By breaking down longer tests into various sub-tests, such as user interface testing, function testing, UX testing and security testing, teams can make a more efficient analysis of each test.
- **Use two-tier test automation.** To cover all bases, organizations should use a two-way approach to software testing. Quick sanity checks on each commit to the source code, followed by extensive regression testing during off hours, is a great option. This way developers get instant feedback on the current portion of the code and can fix it immediately instead of backtracking for errors later down the road.
- **Don't skip regression testing.** Regression testing is one of the most important steps to take before an application can finally move to the production phase and it shouldn't be skipped. Since most of the testing has been done before regression testing, it encourages the validation of the entire application.

Exercise on Chapter Two

2. Describe the term software development

Discuss in details the six major steps in the software development life cycle

i. Needs identification ii. Requirement analysis iii. Design

iv. Development and implementation v. Testing vi. Deployment and maintenance

c). Briefly discuss the three main groups of software list below:

i. System software ii. Application software iii. Programming languages

d). What do you understand by software testing? State why software testing is important to software developer?

e). Explain the following testing activities in software development:

i). Unit testing ii). Integration testing iii). System Testing iv). Acceptance Testing

f). Identify and discuss any five important reasons why software testing techniques should be incorporated into application development