NAME: JUBRIL DAMILOLA

MATRIC NUMBER: 17/SCI01/044

COURSE CODE: CSC308

COURSE TITLE: Formal Methods and Software Development

ASSIGNMENT

1(a)i Describe by using a simple diagram the concept of formal methods.

1(a)ii What are the major reasons for considering formal methods.

1(b) Differentiate between functional and non-functional requirement in formal methods.

2(a) List and discuss briefly the recommended development process of software engineering methodology.

2(b) Using spectrum of rigor, discuss what is meant by formal methods.

3(a) i Differentiate between propositional logic and first order predicate logic.

3(a) ii Using propositions A, B, and C; discuss any three basic logical operations in propositional logic

3(b) Represent the text **“Every man is mortal. Smith is a man. Therefore, Smith is mortal”** in first order predicate calculus expression.

4(a) Using the principles of first order predicate calculus to represent this statement

**“There exists an object that is either a curious monkey or not a monkey at all”**.

4(b) List and discuss any three (3) types of Formal Specification

5(a) Define the following:

(i) A well-formed formula (ii) A quantifier (iii) A Predicate (iv) A Term

5(b) Translate the following Predicate Calculus to statements

(i) GREATER (a, b) = T, if a < b = F, otherwise.

(ii) (ꓯ y) LIKE (Mother (y), y).

6(a) Discuss with examples any three (3) software development strategies

6(b) What do you understand by requirement document

7(a) Discuss the structure of a requirement document

7(b) Enumerate the steps involved in writing a system requirement specification

**Answers**

1(a)i The term “formal methods” refers to various mathematical techniques used for the formal specification and development of software. They consist of a formal specification language and employ a collection of tools to support the syntax checking of the specification, as well as the proof of properties of the specification.



1(a)ii 1.To formally specify a software system, to state its crucial properties and to prove a system correct for all behaviors.And verify the possibility of implementing a property of a system or the system itself, before resources are spent.

1. Testing can show errors but not their absence, but with the formal method we can mathematically proven that there are no errors present in the system. Software errors in critical systems can cause major disasters.
2. Systems are becoming increasingly dependent on software components.
3. Complexity of Systems with embedded software has increased;A modern mid-sized sedan car is equipped with at least 150 processors, making it the very important to employ formal method to reduce the complexity for the development crew.

1(b) Any project’s requirements need to be well thought out, balanced and clearly understood by all involved, but perhaps of most importance is that they are not dropped or compromised halfway through the project.Functional requirements will specify the behaviour or functions that a system should have or do.for example:
“Display the name, total size, available space and format of a flash drive connected to the USB port.”

 While The definition for a non-functional requirement is that it essentially specifies how the system should behave and that it is a constraint upon the systems behaviour. One could also think of non-functional requirements as quality attributes for of a system.

 Simply put, the difference is that non-functional requirements describe how the system works, while functional requirements describe what the system should do.

2(a) Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

**I. Agile software development**

Uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system.There are many agile methodologies namely: dynamic system development method(DSDM), kanban, scrum

**II. Waterfall development**

The waterfall model is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through several phases, typically:

Requirements analysis resulting in a software requirements specification

Software design

Implementation

Testing

Integration, if there are multiple subsystems

Deployment (or Installation)

Maintenance

**III. Spiral development**

This is a methodology which combines some key aspect of the waterfall model and rapid prototyping methodologies, in an effort to combine advantages of top-down and bottom-up concepts. It provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems.

**IV. Incremental funding**

Methodology is an ROI-informed approach to software development in which software is developed and delivered in carefully prioritized chunks of customer valued functionality. These chunks are known as Minimum Marketable Features (MMFs).

 Other software development methodologies include

Prototyping

Behavior-driven development and business process management

Lightweight methodology - a general term for methods that only have a few rules and practices

And so on.

2(b)



Fig. 1: Spectrum of rigor.

Formal methods can assume various forms and levels of rigor. At one extreme is the least rigorous level of rigour while most rigorous level of rigor lies at the other extreme. (Fig. 1).Lightweight formal methods usually do not require deep expertise, by opposition to heavyweight formal methods, which are more complex, less automatic, but also more finely grained and powerful. They are typically confined to very specific application areas where their use and cost are justified.

3(a) Propositional logic is logic that includes sentence letters(A,B,C) and logical connectives, but not quantifiers.The semantics of propositional logic uses truth assignments to the letters to determine whether a compound propositional sentence is true.While predicate logic also know as first order logic has the same connectives as propositional logic but it also has variables for individual objects, Quantifiers, symbols for function and symbol for relations.

3(a) ii **Logical operations in propositional logic**

**Negation:** The simplest logical operation is negation. It is unary. The logical negation of the proposition A, is !A. The operator ! is sometimes represented by the symbol ¬, a minus sign (−), a tilde (˜), or the word "not." The negation of A is sometimes called the inverse of A. Logical negation is like a negative sign in arithmetic (a negative sign, not a minus sign, which operates on a pair of numbers. Truth table below

A B C !A !B !C

0 0 0 1 1 1

0 0 1 1 1 0

0 1 0 1 0 1

0 1 1 1 0 0

1 0 0 0 1 1

1 0 1 0 1 0

1 1 0 0 0 1

1 1 1 0 0 0

**Disjunction (|):** It is an operation on two propositions (a binary operation) that results in another proposition: the proposition (A | B) is true if p is true or if q is true or if both p and q are true. The operation | is sometimes represented by a vee (∨) or by the word "or." The truth table for disjunction is given below:

A B A|B

0 0 0

0 1 1

1 0 1

1 1 1

**Conjunction(&)**: It combines two propositions to produce another. The proposition (p & q) is true if both p is true and q is true; it is false if either p is false or q is false (or both). The operation & is sometimes represented by a wedge (∧) or the word "and." Here is the truth table for &:

A B A&B

0 0 0

0 1 0

1 0 0

1 1 1

3(b) **ꓯ**x (man(x)→mortal(x))

P = man , Q = mortal

**ꓯ**x (P(x)→Q(x))

**ꓯ**x **Ǝ** x (P(x)→Q(x))

4(a) **Ǝ** x : Monkey(x)→Curious(x)

4(b) **History-based specification**

behavior based on system histories

assertions are interpreted over time

**State-based specification**

behavior based on system states

series of sequential steps, (e.g. a financial transaction)

languages such as Z, VDM or B rely on this paradigm

**Transition-based specification**

behavior based on transitions from state-to-state of the system

best used with a reactive system

languages such as Statecharts, PROMELA, STeP-SPL, RSML or SCR rely on this paradigm.

5(a) In mathematical logic, a predicate is commonly understood to be a Boolean-valued function P: X→ {true, false}, called a predicate on X. However, predicates have many different uses and interpretations in mathematics and logic, and their precise definition, meaning and use will vary from theory to theory. So, for example, when a theory defines the concept of a relation, then a predicate is simply the characteristic function (otherwise known as the indicator function) of a relation. However, not all theories have relations, or are founded on set theory, and so one must be careful with the proper definition and semantic interpretation of a predicate.

**A quantifier** is a language element that helps in generation of a quantification, which is a construct that mentions the number of specimens in the given domain of discourse satisfying a given open formula.

**The term** refers to computer-generated image data created with the help of specialized graphical hardware and software. It is a vast and recently developed area of computer science.

**A well-formed formula** is a finite sequence of symbols from a given alphabet that is part of a formal language.

5(b)(i) if "a" is greater than "b" then it is true otherwise false

6(a)i **Waterfall model:**The waterfall model is one of the most traditional and commonly used software development methodologies for software development. This life cycle model is often considered as the classic style of the software development. This model clarifies the software development process in a linear sequential flow that means that any phase in the development process begins only if the earlier phase is completed. This development approach does not define the process to go back to the previous phase to handle changes in requirements.

**Example of Waterfall model**:In the olden days, Waterfall model was used to develop enterprise applications like Customer Relationship Management (CRM) systems, Human Resource Management Systems (HRMS), Supply Chain Management Systems, Inventory Management Systems, Point of Sales (POS) systems for Retail chains etc.

Waterfall model was used significantly in the development of software till the year 2000. Even after the Agile manifesto was published in 2001, Waterfall model continued to be used by many organization till the last decade.

Consider a system where human life is on the line, where a system failure could result in one or more deaths. In such situations, Waterfall model was the preferred approach.

Development of Department Of Defense (DOD), military and aircraft programs followed Waterfall model in many organizations.This is because of the strict standards and requirements that have to be followed.

Waterfall model was also used in banking, healthcare, control system for nuclear facilities, space shuttles etc.

**Prototype model:** The prototype methodology is the software development process which allows developers to create only the prototype of the solution to demonstrate its functionality to the clients and make necessary modifications before developing the actual application. The best feature of this software development methodologies is that it solves many issues which often occur in a traditional waterfall model.

**Example of Prototype model:**There are many systems that have been developed with the prototype model and it achieved success as well for both developers and client. The dynamic system development method, evolutionary system method and rapid method is being developed. There are some tools, screen generators are developed with the prototype model.

**Rapid Application Development:** Rapid Application Development (RAD) is an effective methodology to provide much quicker development and higher-quality results than those achieved with the other software development methodologies. It is designed in such a way that, it easily take the maximum advantages of the software development. The main objective of this methodology is to accelerate the entire software development process. The goal is easily achievable because it allows active user participation in the development process.

**Examples of RAD:**

1. Purchase Order

Collecting data for purchase orders and approving them sounds like a very simple process, but readymade options often complicate it. However, you also want to build them on a platform that gives you more than just basic functionality.

2.Employee Resignation

Another RAD example is handling employee resignation. HR teams have a lot to coordinate when an employee decides to leave the company. This app might seem trickier to build just because there are so many moving parts involved.

6(b)A product requirements document (PRD) is a document containing all the requirements to a certain product. It is written to allow people to understand what a product should do. A PRD should, however, generally avoid anticipating or defining how the product will do it in order to later allow interface designers and engineers to use their expertise to provide the optimal solution to the requirements.[citation needed]

PRDs are most frequently written for software products, but can be used for any type of product and also for services. Typically, a PRD is created from a user's point-of-view by a user/client or a company's marketing department (in the latter case it may also be called Marketing Requirements Document (MRD)). The requirements are then analyzed by a (potential) maker/supplier from a more technical point of view, broken down and detailed in a Functional Specification (sometimes also called Technical Requirements Document).

7a Discuss the structure of a requirement document.

Structure of a Requirements Document

Requirements have to be specified using some specification language. Though formal notations exist for specifying specific properties of the system, natural languages are now most often used for specifying requirements. When formal languages are employed, they are often used to specify particular properties or for specific parts of the system, as part of the overall SRS.

All the requirements for a system, stated using a formal notation or natural language, have to be included in a document that is clear and concise. For this, it is necessary to properly organize the requirements document.

The IEEE standards recognize the fact that different projects may require their requirements to be organized differently, that is, there is no one method that is suitable for all projects. It provides different ways of structuring the SRS. The first two sections of the SRS are the same in all of them.

The introduction section contains the purpose, scope, overview, etc., of the requirements document. The key aspect here is to clarify the motivation and business objectives that are driving this project, and the scope of the project.

The next section gives an overall perspective of the system—how it fits into the larger system, and an overview of all the requirements of this system.

Detailed requirements are not mentioned. Product perspective is essentially the relationship of the product to other products; defining if the product is independent or is a part of a larger product, and what the principal interfaces of the product are. A general abstract description of the functions to be performed by the product is given. Schematic diagrams showing a general view of different functions and their relationships with each other can often be useful. Similarly, typical characteristics of the eventual end user and general constraints are also specified.

If agile methods are being used, this may be sufficient for the initial requirements phase, as these approaches prefer to do the detailed requirements when the requirement is to be implemented.

The external interface requirements section specifies all the interfaces of the software: to people, other software, hardware, and other systems. User interfaces are clearly a very important component; they specify each human interface the system plans to have, including screen formats, contents of menus, and command structure. In hardware interfaces, the logical characteristics of each interface between the software and hardware on which the software can run are specified. Essentially, any assumptions the software is making about the hardware are listed here. In software interfaces, all other software that is needed for this software to run is specified, along with the interfaces. Communication interfaces need to be specified if the software communicates with other entities in other machines.

In the functional requirements section, the functional capabilities of the system are described. In this organization, the functional capabilities for all the modes of operation of the software are given. For each functional requirement, the required inputs, desired outputs, and processing requirements will have to be specified. For the inputs, the source of the inputs, the units of measure, valid ranges, accuracies, etc., have to be specified. For specifying the processing, all operations that need to be performed on the input data and any intermediate data produced should be specified. This includes validity checks on inputs, sequence of operations, responses to abnormal situations, and methods that must be used in processing to transform the inputs into corresponding outputs.

The performance section should specify both static and dynamic performance requirements. All factors that constrain the system design are described in the performance constraints section. The attributes

section specifies some of the overall attributes that the system should have. Any requirement not covered under these is listed under other requirements. Design constraints specify all the constraints imposed on design (e.g., security, fault tolerance, and standards compliance).

When use cases are employed, then the functional requirements section of the SRS is replaced by use case descriptions. And the product perspective part of the SRS may provide an overview or summary of the use cases.

7b)

1. Create an outline

An outline is a plan of your document that contains key points about the product and its scope. You can use it for generating new ideas and organize them in a logical structure. Here’s a sample of an outline.

1. Introduction

1.1. Purpose

1.2. Document Conventions

1.3. Intended Audience

1.4. Product Scope

1.5. References

2. Overall Description

2.1. Product Perspective

2.2. Product Functions

2.3. User Classes and Characteristics

2.4. Operating Environment

2.5. Design and Implementation Constraints

2.6. User Documentation

2.7. Assumptions and Dependencies

3. External Interface Requirements

3.1. User Interfaces

3.2. Hardware Interfaces

3.3. Software Interfaces

3.4. Communications Interfaces

4. System Features

4.1. System Feature 1

4.2. System Feature 2 (and so on)

5. Other Non-Functional Requirements

5.1. Performance Requirements

5.2. Safety Requirements

5.3. Security Requirements

5.4. Software Quality Attributes

5.5. Business Rules

6. Other Requirements

Appendix.

2. Define the goals

To define the goals of your project, answer the following questions:

**What is a new product developed for?**

**What kind of value does it bring?**

**Who are the major competitors? Can you make a better product?**

**What are the essential requirements for a future project?**

You can sum up the answers and formulate the objectives. It’s good to make a list of people that will have access to the document and how they will use it. You can add BAs, project managers, coders, QAs, people from the client’s side and your sales (marketing) teams.

**Use and Audience**

During this stage of the SRS writing process, you should decide on who will be able to read and use the SRS documentation. The intended group of people may include test engineers, developers, PMs, and other stakeholders.

**Scope and definitions**

The scope should include all the benefits the stakeholders can get with the software under development. It must also describe the general business goals. Add the definitions of the terms you use throughout the document.

3. Add details to the requirements

There are a few types of requirements that you should add to the document. We can roughly divide them into high-level and low-level ones. Here are the high-level requirements:

Business requirements – goals, mission, problems to solve, etc.

User needs – the interaction of a user with a new system, functionalities they need.

System requirements – functional and non-functional.

Here are the essential low-level requirements:

a. User interface

b.Market specs

c.Stakeholder specs

d.Let’s take a closer look at system specifications.

4. Get an SRS document approved

You should show the document to the people that are participating in the development process. You can make a short presentation for the stakeholders if it’s required. You may have the need to update the document or add any other important issues.

This SRS example will help you structure your own specifications. Besides, we’ve collected useful tips for writing specs.