Name: Ibekwe Brian Ogochukwu

Matric: 17/sci01/037

Level: 300

**Assignment Title:** Formal Methods and Software Development Assignment  
**Course Title:** Formal Methods and Software Development  
**Course Code:** CSC 308

**Question**

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

**Solution**

1b We consider formal methods because

**•** Systems are increasingly dependent on software components

**•** Complexity of systems with embedded software has increased rapidly

**•** Maintenance reliability in software intensive systems is very difficult

1c

Nonfunctional requirements describe the general characteristics of a system. They are also

known as quality attributes. Functional requirements describe how a product must behave, what

its features and functions.

2a

Agile development methodology

Teams use the agile development methodology to minimize risk (such as bugs, cost overruns,

and changing requirements) when adding new functionality. In all agile methods, teams

develop the software in iterations that contain mini-increments of the new functionality. There

are many different forms of the agile development method, including scrum, crystal, extreme

programming (XP), and feature-driven development (FDD).

Waterfall development method

Many consider the waterfall method to be the most traditional software development method.

The waterfall method is a rigid linear model that consists of sequential phases (requirements,

design, implementation, verification, maintenance) focusing on distinct goals. Each phase must

be 100% complete before the next phase can start. There’s usually no process for going back to

modify the project or direction.

Rapid application development

Rapid application development (RAD) is a condensed development process that produces

a high-quality system with low investment costs. Scott Stiner, CEO and president of UM

Technologies, said in Forbes, “This RAD process allows our developers to quickly adjust to

shifting requirements in a fast-paced and constantly changing market.” The ability to quickly

adjust is what allows such a low investment cost.

The rapid application development method contains four phases: requirements planning, user

design, construction, and cutover. The user design and construction phases repeat until the user

confirms that the product meets all requirements.

DevOps deployment methodology

DevOps is not just a development methodology but also a set of practices that supports an

organizational culture. DevOps deployment centers on organizational change that enhances

collaboration between the departments responsible for different segments of the development

life cycle, such as development, quality assurance, and operations.

6b

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 softwareproducts, 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).

7b

1. Introduction

1.1 Purpose

1.2 Intended Audience

1.3 Intended Use

1.4 Scope

1.5 Definitions and Acronyms

2. Overall Description

2.1 User Needs

2.2 Assumptions and Dependencies

3. System Features and Requirements

3.1 Functional Requirements

3.2 External Interface Requirements

3.3 System Features

3.4 Nonfunctional Requirements

Once you have your basic outline, you’re ready to start filling it out.

2. Start With a Purpose

The introduction to your SRS is very important. It sets the expectation for the product you’re

building.

So, start by defining the purpose of your product.

Intended Audience and Intended Use

Define who in your organization will have access to the SRS — and how they should use it. This

may include developers, testers, and project managers. It could also include stakeholders in

other departments, including leadership teams, sales, and marketing.

Product Scope

Describe the software being specified. And include benefits, objectives, and goals. This should

relate to overall business goals, especially if teams outside of development will have access to

the SRS.

Definitions and Acronyms

It’s smart to include a risk definition. Avoiding risk is top-of-mind for many developers

— especially those working on safety-critical development teams.

Here’s an example. If you’re creating a medical device, the risk might be the device fails and

causes a fatality.

By defining that risk up front, it’s easier to determine the specific requirements you’ll need to

mitigate it.

3. Give an Overview of What You’ll Build

Your next step is to give a description of what you’re going to build. Is it an update to an existing

product? Is it a new product? Is it an add-on to a product you’ve already created?

These are important to describe upfront, so everyone knows what you’re building.

You should also describe why you’re building it and who it’s for.

User Needs

User needs — or user classes and characteristics — are critical. You’ll need to define who is

going to use the product and how.

You’ll have primary and secondary users who will use the product on a regular basis. You may

also need to define the needs of a separate buyer of the product (who may not be a primary/

secondary user). And, for example, if you’re building a medical device, you’ll need to describe the

patient’s needs.

Assumptions and Dependencies

There might be factors that impact your ability to fulfill the requirements outlined in your SRS.

What are those factors?

Are there any assumptions you’re making with the SRS that could turn out to be false? You

should include those here, as well.

Finally, you should note if your project is dependent on any external factors. This might include

software components you’re reusing from another project.

4. Detail Your Specific Requirements

The next section is key for your development team. This is where you detail the specific

requirements for building your product.

Functional Requirements

Functional requirements are essential to building your product.

If you’re developing a medical device, these requirements may include infusion and battery. And

within these functional requirements, you may have a subset of risks and requirements.

External Interface Requirements

External interface requirements are types of functional requirements. They’re important for

embedded systems. And they outline how your product will interface with other components.

There are several types of interfaces you may have requirements for, including:

User

Hardware

Software

Communications

System Features

System features are types of functional requirements. These are features that are required in

order for a system to function.

Other Nonfunctional Requirements

Nonfunctional requirements can be just as important as functional ones.

These include:

Performance

Safety

Security

Quality

The importance of this type of requirement may vary depending on your industry. Safety

requirements, for example, will be critical in the medical device industry.

IEEE also provides guidance for writing software requirements specifications, if you’re a

member.

5. Get Approval for the SRS

Once you’ve completed the SRS, you’ll need to get it approved by key stakeholders. And

everyone should be reviewing the latest version of the document.

6a

Build-and-Fix Model

The build-and-fix model was adopted from an earlier and simpler age of hardware product

development. Those of us who bought early Volkswagen automobiles in the 1950s and ’60s

remember it well. As new models were brought out and old models updated, the cars were

sold apparently without benefit of testing, only to be tested by the customer. In every case,

the vehicles were promptly and cheerfully repaired by the dealer at no cost to their owners,

except for the inconvenience and occasional risk of a breakdown. This method clearly works,

but it depends on having a faithful and patient customer set almost totally dependent on the

use of your product! It is the same with software. A few well-known vendors are famous for

their numerous free upgrades and the rapid proliferation of new versions. This always works

best in a monopolistic or semimonopolistic environment, in which the customer has limited

access to alternative vendors. Unfortunately in the build-and-fix approach, the product’s overall

quality is never really addressed, even though some of the development issues are ultimately

corrected. Also, there is no way to feed back to the design process any proactive improvement

approaches. Corrections are put back into the market as bug fixes, service packs, or upgrades

as soon as possible as a means of marketing "damage control." Thus, little learning takes place

within the development process. Because of this, build-and-fix is totally reactive and, by today’s

standards, is not really a development model at all

Rapid Prototyping Model

Rapid prototyping has long been used in the development of one-off programs, based on the

familiar model of the chemical engineer’s pilot plant. More recently it has been used to prototype

larger systems in two variants—the "throwaway" model and the "operational" model, which

is really the incremental model to be discussed later. This development process produces a

program that performs some essential or perhaps typical set of functions for the final product.

A throwaway prototype approach is often used if the goal is to test the implementation method,

language, or end-user acceptability. If this technology is completely viable, the prototype may

become the basis of the final product development, but normally it is merely a vehicle to arrive at

a completely secure functional specification.

Incremental Model

The incremental model recognizes that software development steps are not discrete. Instead,

Build 0 (a prototype) is improved and functionality is added until it becomes Build 1, which

becomes Build 2, and so on. These builds are not the versions released to the public but

are merely staged compilations of the developing system at a new level of functionality or

completeness. As a major system nears completion, the project manager may schedule a new

build every day at 5 p.m. Heaven help the programmer or team who does not have their module

ready for the build or whose module causes compilation or regression testing to fail!

5a

Well formed formula: Well-formed formula. In mathematical logic, propositional logic and

predicate logic, awell-formed formula, abbreviated WFF or wff, often simply formula, is a finite

sequence of symbols from a given alphabet that is part of a formal language. A formal language

can be identified with the set of formulas in the language.

Quantifiers: Quantifiers are expressions or phrases that indicate the number of objects

that a statement pertains to. There are two quantifiers in mathematical logic: existential

and universal quantifiers. ... ' Some words and phrases in a statement that indicate an

existentialquantifier are 'some,' 'at least one,' and 'there is.

A predicate: state, affirm, or assert (something) about the subject of a sentence or an argument

of a proposition

4b

Algebraic Specification

The use of modularization, datatypes, and object oriented programming have led to a further

model called algebraic specifications, as developed by Guttag. In this model we are more

concerned about the behavior of objects defined by programs rather than the details of their

implementation

Z specification

The Z notation is a formal specification language used for describing and modelling computing

systems. It is targeted at the clear specification of computer programs and computer-based

systems in general.

Model-Based Languages

One approach to formal specifications is to build a model of the intended system by describing

the different states the system could be in and the operations that will change

the state. The states are often described with sets, sequences, relations, and functions, and

the operations with predicates in terms of pre- and post-conditions.

2b

They are techniques and tools based on mathematical and formal logic and can achieve various

forms of vigour