NAME: UYE TIMILEYIN

COURSE CODE: CSC 308

MATRIC NO: 17/SCI01/081

**QUESTIONS**

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

**SOLUTIONS**

**QUESTION 1**

**1a)** Diagram explaining the concept of formal methods

 

**1a(ii**) There are strong motivations to use best practice in software engineering in order to produce software adhering to high-quality standards. Some of these motivations are:

- Systems are becoming increasingly dependent on software components;

- Complexity of Systems with embedded software has increased - A modern mid-sized sedan car is equipped with at least 150 processors;

- Maintaining reliability in software-intensive systems is very difficult. Quality problems with software may cause minor irritations or major damage to a customer’s business including loss of life.

1b) **Functional requirements** consists of the statement of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations and it may also state what the system should not do while **Non-functional requirement** consists of constraints on the services or functions offered by the system such as timing constraints, constraints on development process, standards etc. It also applies to the system as a whole rather than an individual features or services.

**QUESTION 2**

**Prototyping:**

 Software prototyping is about creating prototypes, i.e. incomplete versions of the software program being developed.

The basic principles are:

* Prototyping is not a standalone, complete development methodology, but rather an approach to try out particular features in the context of a full methodology (such as incremental, spiral, or rapid application development (RAD)).
* Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
* The client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.
* While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problems, but this is true for all software methodologies.

**Incremental development** :

 Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

There are three main variants of incremental development:

1. A series of mini-Waterfalls are performed, where all phases of the Waterfall are completed for a small part of a system, before proceeding to the next increment, or
2. Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
3. The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by incremental implementation, which culminates in installing the final version, a working system.

**Rapid application development**



Rapid Application Development (RAD) Model

 (RAD) is a software development methodology, which favors iterative developments and the rapid construction of prototypes instead of large amounts of up-front planning. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

The rapid development process starts with the development of preliminary data models and business process models using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems.

The basic principles of rapid application development are:

* Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.
* Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
* Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders,  Computer Aided Software Engineering(CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
* Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
* Project control involves prioritizing development and defining delivery deadlines or “timeboxes”. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the
* Generally includes joint application designs (JAD), where users are intensely involved in system design, via consensus building in either structured workshops, or electronically facilitated interaction.
* Active user involvement is imperative.
* Iteratively produces production software, as opposed to a throwaway prototype.
* Produces documentation necessary to facilitate future development and maintenance.
* Standard systems analysis and design methods can be fitted into this framework.

**Waterfall development**[



The activities of the software development process represented in the water fall model. There are several other models to represent this process.

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

* Requirement analysis resulting in a software requirement specification
* Software design
* Implementation
* Testing
* Integration, if there are multiple subsystems
* Deployment (or installation)
* Maintenance

The basic principles are:[[1]](https://en.wikipedia.org/wiki/Software_development_process%22%20%5Cl%20%22cite_note-CMS08-1)

* Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
* Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
* Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase. Written documentation is an explicit deliverable of each phase.

The waterfall model is a traditional engineering approach applied to software engineering. A strict waterfall approach discourages revisiting and revising any prior phase once it is complete. This "inflexibility" in a pure waterfall model has been a source of criticism by supporters of other more "flexible" models. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on requirements due to the big design up front approach. Except when contractually required, the waterfall model has been largely superseded by more flexible and versatile methodologies developed specifically for software development.

**Spiral development**



Spiral model (Boehm, 1988)

The spiral method combines some key aspect of the waterfall method 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.

The basic principles are:

* Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
* "Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program."
* Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.
* Begin each cycle with an identification of stakeholders and their "win conditions", and end each cycle with review and commitment.

2b) Formal methods can assume various forms and levels of rigor. At one extreme is the least rigorous level of rigor while most rigorous level of rigor lies at the other extreme. An interesting characterization used to classify formal methods is the one that takes into account the ease of use and automation of the processes involved in their application. This gives rise to the classification of lightweight versus heavyweight usually found in the literature. 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.

**QUESTION 3**

3ai) The key difference between propositional logic and first order predicate is that Propositional Logic converts a complete sentence into a symbol and makes it logical whereas in **First**-**Order Logic** relation of a particular sentence will be made that involves relations, constants, functions, and constants.

3aii) Conjunction: The And (∧) is the [truth-functional](https://en.wikipedia.org/wiki/Truth_function) operator of **logical conjunction**; the *and* of a set of operands is true if and only if *all* of its operands are true. The [logical connective](https://en.wikipedia.org/wiki/Logical_connective) that represents this operator is typically written as ∧ or ⋅ .

Implication: **Logical implication is** a type of relationship between two statements or sentences. ... If A and B represent statements, then A B **means** "A implies B" or "If A, then B." The word "implies" **is** used in the strongest possible sense.

Tautology: A **tautology** is a formula which is "always true" --- that is, it is true for every assignment of truth values to its simple components. You can think of a tautology as a *rule of logic*.The opposite of a tautology is a **contradiction**, a formula which is "always false". In other words, a contradiction is false for every assignment of truth values to its simple components.

{\displaystyle A\land B}{\displaystyle A}{\displaystyle B}3b)m1=mortal

m2=man

P1=every man is mortal

P2=smith is a man

[Smith is mortal]

 ∀(m2):m1(p1,p2)

**QUESTION 4**

a)o = object

m = monkey

~m = not monkey

∃(o): m v ~m

b) **Model-based specification** is an approach to formal specification where the system specification is expressed as a system state model. This state model is constructed using well-understood mathematical entities such as sets and functions. System operations are specified by defining how they affect the state of the system model.

The most widely used notations for developing model-based specifications are VDM and Z (pronounced Zed, not Zee). These notations are based on typed  Set theory. Systems are therefore modeled using sets and relations between sets.

**Algebraic specification** is a software engineering technique for formally specifying system behavior. Algebraic specification seeks to systematically develop more efficient programs by:

1. formally defining types of data, and mathematical operations on those data types
2. abstracting implementation details, such as the size of representations (in memory) and the efficiency of obtaining outcome of computations
3. formalizing the computations and operations on data types
4. Allowing for automation by formally restricting operations to this limited set of behaviors and data types.

An algebraic specification achieves these goals by defining one or more data types, and specifying a collection of functions that operate on those data types. These functions can be divided into two classes:

1. constructor function: functions that create or initialize the data elements, or construct complex elements from simpler ones
2. Additional function: functions that operate on the data types, and are defined in terms of the constructor functions.

**Transition System based Specifications** (TSSs) [16] are a formalisation of Structural Operational Semantics [22] providing programming and **specification** languages with an interpretation. They provide the mean- ing of a closed term as a process graph: a state in a labelled **transition** system.

**QUESTION 5**

5ai) Well defined formula: A well-formed formula (WFF) in propositional logic is a syntactically correct formula created according to the syntactic rules of the underlying calculus. A well-formed formula is built up from variables, constants, terms and logical connectives such as conjunction (and), disjunction (or), implication (if … then …), equivalence (if and only if) and negation. The disjunction operator is known as the “inclusive or” operator as it is also true when both A and B are true; there is also an exclusive or operator that is true exactly when one of A or B is true, and is false otherwise. Distinguished subsets of these well-formed formulae are the axioms of the calculus, and there are rules of inference that allow the truth of new formulae to be derived from the axioms and from formulae that have already demonstrated to be true in the calculus.

ii)Quantifier: A quantifier declares the scope or variables in a logical expression. There are two basic quantifiers which are Universal quantifier and Existential quantifier.  The phrase "for every xx'' (sometimes "for all xx'') is called a **universal quantifier** and is denoted by ∀x∀x. The phrase "there exists an xx such that'' is called an **existential quantifier** and is denoted by ∃x∃x. A formula that contains variables is not simply true or false unless each of these variables is **bound** by a quantifier. If a variable is not bound the truth of the formula is contingent on the value assigned to the variable from the universe of discourse. To define the truth values of compound statements precisely. We do the same for ∀xP(x)∀xP(x) and ∃xP(x)∃xP(x), though the intended meanings of these are clear.

iii) A predicate: 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. Informally, a predicate is a statement that may be true or false depending on the values of its variables. It can be thought of as an operator or function that returns a value that is either true or false For example, predicates are sometimes used to indicate set membership: when talking about sets, it is sometimes inconvenient or impossible to describe a set by listing all of its elements. Thus, a predicate *P(x)* will be true or false, depending on whether *x* belongs to a set.

iv) A term: a **term** denotes a mathematical object and a formula denotes a mathematical fact. In particular, terms appear as components of a formula.

5b)

**QUESTION 6**

#### 6a) Problem definition

“What are the current problems” This stage of the SDLC means getting input from all stakeholders, including customers, salespeople, industry experts, and programmers. Learn the strengths and weaknesses of the current system with improvement as the goal.

#### 2. Plan

“What do we want?” In this stage of the SDLC, the team determines the cost and resources required for implementing the analyzed requirements. It also details the risks involved and provides sub-plans for softening those risks.

In other words, the team should determine the feasibility of the project and how they can implement the project successfully with the lowest risk in mind.

#### 3. Design

“How will we get what we want?” This phase of the SDLC starts by turning the software specifications into a design plan called the Design Specification. All stakeholders then review this plan and offer feedback and suggestions. It’s crucial to have a plan for collecting and incorporating stakeholder input into this document. Failure at this stage will almost certainly result in cost overruns at best and the total collapse of the project at worst.

b) Requirement Document : Are also known as product requirement 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. PRDs 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.

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 requirement 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).

**QUESTION 7**

7a)General structure of an SRS

 

 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 detailed requirements section describes the details of the requirements that a developer needs to know for designing and developing the system. This is typically the largest and most important part of the document. For this section, different organizations have been suggested in the standard. These requirements can be organized by the modes of operation, user class, object, feature, stimulus, or functional hierarchy [53]. One method to organize the specific requirements is to first specify the external interfaces, followed by functional

requirements, performance requirements, design constraints, and system attributes. This structure is shown in Figure

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.



Fig: one requirement for specific requirement

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) The **four basic process** activities of **specification**, development, validation, and evolution are organized differently in different development processes.

1. Feasibility study :An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more The requirements engineering process detailed analysis (feasibility report).

 2. Requirements elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and buyer, task analysis. This may involve the development of one or more system models and prototypes. These help the system developer understand the system to be specified.

3. Requirements specification: Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and endorser of the system; System requirements are a more detailed description of the functionality to be provided.

 4. Requirements validation: This activity checks the requirements for realism , consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems. Of course, the activities in the requirements process are not simply carried out in a strict sequence. Requirements analysis continues during definition and specification and new requirements come to light throughout the process. Therefore, the activities of analysis, definition, and specification are interleaved. In agile methods, such as Extreme Programming, requirements are developed incrementally according to user priorities and the elicitation of requirements comes from users who are part of the development team.