**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

1. 

2.

a. Measure of correctness: The use of formal methods provides a measure of the correctness of a system, as opposed to the current process quality measures.

b. Early defect detection: Formal Methods can be applied to the earliest design artifacts, thereby leading to earlier detection and elimination of design defects.

c. Guarantees of correctness: Formal analysis tools such as model checkers consider all possible execution paths through the system. If there is any possibility of a fault/error, a model checker will find it. In a multithreaded system where concurrency is an issue, formal analysis can explore all possible interleaving and event orderings. This level of coverage is impossible to achieve through testing.

d. Error Prone: Formal description forces the writer to ask all sorts of questions that would otherwise be postponed until coding. This helps to reduce the errors that occur during or after coding. Formal methods have the property of completeness, i.e. it covers all aspects of the system.

e. Abstraction: If the working of software or hardware product is simple, then one can write the code straight away, but in the majority of systems the code is far too big, which generally needed the detailed description of the system. A formal specification, on the other hand, is a description that is abstract, precise and in some senses complete. The abstraction allows a human reader to understand the big picture of the software product easily.

f. Rigorous Analysis: The formality of the description allows us to carry out rigorous analysis. Formal descriptions are generally written from different points of view, by which one can determine important properties such as satisfaction of high level requirements or correctness of a proposed design.

g. Trustworthy: Formal methods provide the kind of evidence that is needed in heavily regulated industries such as aviation. They demonstrate and provide concrete reasons for the trust in the product.

h. Effective Test Cases: From formal specification, we can systematically derive effective test cases directly from the specification. It’s a cost effective way to generate test cases.

## 3. KEY DIFFERENCE

* A functional requirement defines a system or its component whereas a non-functional requirement defines the performance attribute of a software system.
* Functional requirements along with requirement analysis help identify missing requirements while the advantage of Non-functional requirement is that it helps you to ensure good user experience and ease of operating the software.
* Functional Requirement is a verb while Non-Functional Requirement is an attribute
* Types of Non-functional requirement are Scalability Capacity, Availability, Reliability, Recoverability, Data Integrity, etc. whereas transaction corrections, adjustments, and cancellations, Business Rules, Certification Requirements, Reporting Requirements, Administrative functions, Authorization levels, Audit Tracking, External Interfaces, Historical Data management, Legal or Regulatory Requirements are various types of functional requirements.

## 4. 1. WATERFALL

When it comes to software development, Waterfall is the most traditional and sequential choice. Although it’s usually viewed as an ”old school” or outdated method, it’s helpful to understand the history and structure of Waterfall to better appreciate the flexibility of more modern methodologies. First created in 1970, Waterfall was one of the most prominent methodologies for several decades because of its plan-driven approach.

Waterfall requires plenty of structure and documentation up front. It is divided into self-contained stages or steps. The first stage is vital, requiring a full understanding by both developers and customers of the project’s demands and scope before anything begins. The stages are relatively rigid and often follow this sequence: determine the project’s requirements and scope, analyze those requirements, design, implement, test, deploy and finally, maintain.

There’s a lack of flexibility with this approach, meaning what is decided by the customer and developer at the beginning must be seen through. Should any changes need to be made or mistakes addressed toward the end stages, the Waterfall method generally requires a full restart.

Typically, one stage must be finished before the next can begin, which can help with organization and assignments. And because the full scope of the project is understood in advance, software progress can easily be measured. Waterfall is often utilized by large, plan-driven teams who have a very clear understanding of the project’s scope;—however, development teams who don’t operate in a vacuum will likely find better results with the flexibility and agility of more modern methodologies.

## 2. FEATURE-DRIVEN DEVELOPMENT

An iterative and incremental approach to software development, Feature-Driven Development

(FDD) is derived from the Agile methodology and is considered one way to implement it. Similar to Waterfall, FDD is typically viewed as an older methodology, a sort of precursor to modern Lean/Agile implementations. FDD still focuses on the goal of delivering working software frequently and is an especially client-centric approach, making it a good fit for smaller development teams.

Features are a foundational piece of FDD. Features are client-valued pieces of work that, according to the FDD approach, should be delivered every two weeks.

To produce tangible software often and efficiently, FDD has five steps, the first of which is to develop an overall model. Next, build a feature list and then plan by each feature. The final two steps—design by feature and build by feature—will take up the majority of the effort. At each step, status reporting is encouraged and helps to track progress, results, and possible errors. Although efficient response to change is one of FDD’s better attributes, an understanding of the client’s requirements and the overall model at the beginning of the project can reduce any surprises during development.

Additionally, any feature that takes longer than two weeks to design and build must be further broken down into separate features until it meets the two-week rule. The rigid structure of FDD make it less desirable to teams who balance project-driven and break-fix types of work.

## 3. AGILE

The Agile methodology was developed as a response to growing frustrations with Waterfall and other highly structured, inflexible methodologies. This approach is designed to accommodate change and the need to produce software faster.

Agile values individuals and their relationships and interactions over tools; it features customer collaboration throughout the development process; it responds to change instead of following a set-in-stone plan; and it focuses on presenting working software, rather than documentation.

Unlike Waterfall, Agile is well equipped to handle the complexity and variability involved in development projects.Using the Agile approach, teams develop in short sprints or iterations, each of which includes a defined duration and list of deliverables, but in no particular order. During sprints, teams work towards the goal of delivering working software (or some other tangible, testable output).

Agile is collaboration-heavy, focusing on team strengths and efficiency, along with internal feedback from various departments and clients. Client satisfaction is the highest priority with the Agile approach, which teams achieve by continuously delivering working, tested, prioritized features.

## 4. SCRUM

Another way to implement the Agile approach, Scrum borrows from Agile’s foundational beliefs and philosophy that teams and developers should collaborate heavily and daily.

With Scrum, software is developed using an iterative approach in which the team is front and center—experienced and disciplined workers on smaller teams might find the most success with this method, as it requires self-organization and self-management.

Team members break down end goals into smaller goals at the beginning and work through them using fixed-length iterations—or sprints—to build software and showcase it often (which usually last two weeks). Meetings play an important role in the Scrum approach, and during each sprint, daily planning meetings and demos take place to follow progress and gather feedback. This incremental method promotes quick changes and development and adds value to complex projects. Scrum incorporates the structure and discipline of more traditional software development methodologies with the flexibility and iterative practices of modern Agile.

## 5. EXTREME PROGRAMMING

Another Agile framework, Extreme Programming (or XP) focuses on producing higher quality software using the best practices in software development. As with most Agile approaches, XP allows for frequent releases in short development sprints that encourage change when needed.

In general, XP follows a set of values, rather than steps, including simplicity (develop what is required, nothing more); communication (teams must collaborate and work together on every piece of the software); consistent feedback; and respect.

Extreme Programing requires developers to first plan and understand the customer’s user stories—their informal descriptions of certain features. Other practices include: scheduling and dividing work into iterations. Design with simplicity in mind, code and test often, which helps to create fault-free software. Listen to feedback to best understand the functionality, and then test more.

5. The field of Participatory Design (PD) has greatly diversified and we see a broad spectrum of approaches and methodologies emerging. However, to foster its role in designing future interactive technologies, a discussion about accountability and rigour across this spectrum is needed. Rejecting the traditional, positivistic framework, we take inspiration from related fields such as Design Research and Action Research to develop interpretations of these concepts that are rooted in PD׳s own belief system. We argue that unlike in other fields, accountability and rigour are nuanced concepts that are delivered through debate, critique and reflection. A key prerequisite for having such debates is the availability of a language that allows designers, researchers and practitioners to construct solid arguments about the appropriateness of their stances, choices and judgements. To this end, we propose a "tool-to-think-with" that provides such a language by guiding designers, researchers and practitioners through a process of systematic reflection and critical analysis. The tool proposes four lenses to critically reflect on the nature of a PD effort: *epistemology*, *values*, *stakeholders* and *outcomes*. In a subsequent step, the *coherence* between the revealed features is analysed and shows whether they pull the project in the same direction or work against each other. Regardless of the flavour of PD, we argue that this *coherence* of features indicates the level of internal rigour of PD work and that the process of reflection and analysis provides the language to argue for it. We envision our tool to be useful at all stages of PD work: in the planning phase, as part of a reflective practice during the work, and as a means to construct knowledge and advance the field after the fact. We ground our theoretical discussions in a specific PD experience, the ECHOES project, to motivate the tool and to illustrate its workings.

6. **Key differences between PL and FOL**

* 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.
* The limitation of PL is that it does not represent any individual entities whereas FOL can easily represent the individual establishment that means if you are writing a single sentence then it can be easily represented in FOL.
* PL does not signify or express the generalization, specialization or pattern for example ‘QUANTIFIERS’ cannot be used in PL but in FOL users can easily use quantifiers as it does express the generalization, specialization, and pattern.

7.