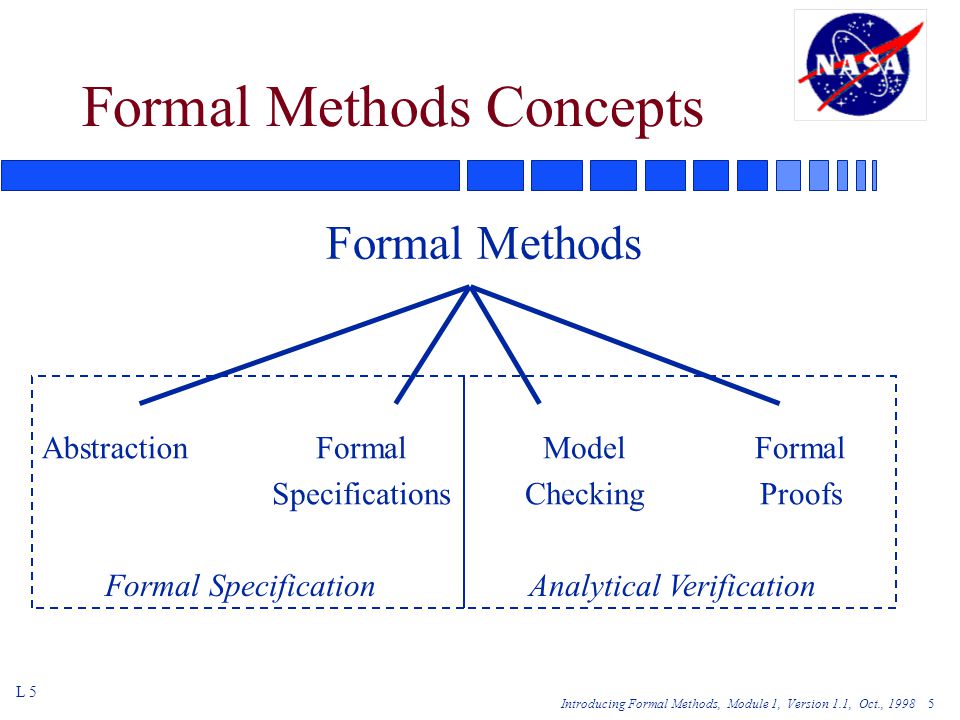
**ITELIMA FAITH IBIFUBARA**

**18/SCI01/106**

**CSC 308**

**1a) i)**

****

**Abstraction:** Simplifies and ignores irrelevant details. It focuses on and generalizes important central properties and characteristics. Avoids premature commitment to design and implementation choices.

**Formal Specifications:** Translates non-mathematical descriptions (diagrams, tables, English text) into a formal specification language. Concise description of high-level behavior and properties of a system. Well-defined language semantics support formal deduction about specification.

**Model Checking:** Operational rather than analytic. State machine model of a system is expressed in a suitable language. Model checker determines if the given finite state machine model satisfies requirements expressed as formulas in a given logic. Basic method is to explore all reachable paths in a computational tree derived from the state machine model.

**Formal Proofs:** Complete and convincing argument for validity of some property of the system description. Constructed as a series of steps, each of which is justified from a small set of rules. Eliminates ambiguity and subjectivity inherent when drawing informal conclusions. May be manual but usually constructed with automated assistance.

**1a)** **ii)** Major reasons for considering formal methods:

* Systems are increasingly dependent on software components.
* Complexity of systems with embedded software has increased rapidly.
* Maintaining reliability in software-intensive systems is very difficult.

**1b)** The official definition of **‘a functional requirement’** is that it essentially specifies something the system should do. Typically, functional requirements will specify a behaviour or function, for example: “Display the name, total size, available space and format of a flash drive connected to the USB port.” Other examples are “add customer” and “print invoice”. Functional requirements refer to services (functionalities) that the system should provide. In order to verify them, one should check if the service (as stated in the specification) is offered or not by the actual system.

Simply put, the difference is that **non-functional requirements** describe how the system works, while functional requirements describe what the system should do. Non-functional requirements refer to constraints & behavioral properties of the system (like efficiency, portability, security, reliability, usability, etc.). For each of them a metric should be associated, and an acceptability level. In order to verify them, one should check if the constraint are satisfied & if the level of satisfaction of the properties is greater than the acceptability level (with respect to the given metrics). 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. Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than specific behaviours, for example: “Modified data in a database should be updated for all users accessing it within 2 seconds.”

**2a)**

* **Waterfall development:** The Waterfall model is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance. It is a linear framework.
* **Prototyping:** Software prototyping, is the development approach of activities during software development, the creation of prototypes, i.e., incomplete versions of the software program being developed. It is an iterative framework.
* **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. It is a combined linear-iterative framework.
* **Spiral development:** The spiral model is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. It is a combined linear-iterative framework.
* **Rapid application development(RAD):** Rapid application development (RAD) is a software development methodology, which involves iterative development and the construction of prototypes. It is an iterative framework.

**2b)** Formal methods can assume various forms and levels of rigor

**Least rigorous** **Spectrum of rigor**  **Most rigorous**

Occasional mathematical notation embedded in English specifications

Fully formal specification languages with a precise semantics

**…**

**3a) I)** Propositional logic is an analytical statement which is either true or false. It is basically a technique that represents the knowledge in logical & mathematical form. It is concerned with the subset of declarative sentences that can be classified as true or false. These sentences are called “statements” or “propositions”. it is comprised of objects, relations, functions, and logical connectives. Proposition logic can either be true or false, it can never be both.

First-Order predicate Logic is another knowledge representation in AI which is an extended part of PL. FOL articulates the natural language statements briefly. In this logic, we call the claim a predicate and we say that we are predicating some property of the object. Unlike propositional logic, first-order predicate logic assumes some of the facts that are related to objects, relations, and functions.

Key differences:

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

**3a) ii) The logical operation &, also called logical conjunction**, combines at least two propositions to produce another. The logical operator & is analogous to multiplication in arithmetic. 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): (T ∧ T) = T, (T ∧ F) = F, (F∧ T) = F, (F ∧ F) = F. The operation & is sometimes represented by a wedge (∧) or the word "and".

**The logical operation |, also called or and logical disjunction**, is an operation on two propositions (a binary operation) that results in another proposition: the proposition (p | q) 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." We can say what | does by saying what value it takes for each of the four combinations of true and false its arguments can have: (T | T) = T, (T | F) = T, (F | T) = T, (F | F) = F.

**The simplest logical operation is negation**. Negation operates on a single proposition—it is unary. The logical negation of the proposition p, is !p. The operator ! is sometimes represented by the symbol ¬, a minus sign (−), a tilde (˜), or the word "not." The negation of p is sometimes called the inverse of p. If p is a proposition, so is !p: !p is true when p is false, and !p is false when p is true. Another way to state this relation is !T = F, and !F = T. Logical negation is like a negative sign in arithmetic (a negative sign, not a minus sign, which operates on a pair of numbers). This can be summarized in a truth table, which displays in tabular form the value for each combination of values of **A,B,C:**

**TRUTH TABLE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **A∧B** | **B|C** | **¬C** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

**3b)** a1=mortal

a2=man

P1=every man is mortal

P2=smith is a man

{smith is mortal}

∀(a2): a1 (p1, p2)

****

**4a)** t = object

m= monkey

~m= not monkey

Ǝ(t): m v ~m

**4b) I) Property oriented:** States desired properties in a purely declarative way. A) Algebraic: data type viewed as an algebra, axioms state properties of data type’s operations. B) Axiomatic: uses first order predicate logic , pre and post combinations operational specification. Describes desired behaviour by providing model of the system.

Ii) **Model oriented:** Provides a direct way of describing system behaviour(sets, sequences, tuples, maps). Abstract model(in terms previously defined mathematical objects e.g, sets, sequences, functions, mappings).

Iii) **Behavioral specifications:** describes constraints on behaviour of implementation e.g., functionality, safety, security, performance.

**5a)**

1. **A well-formed formula:** This 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(iff) and negation(¬).
2. **A Quantifier:** Quantifiers are words, expressions, or phrases that indicate the number of elements that a statement pertains to. In mathematical logic, there are two quantifiers: 'there exists' and 'for all'.
3. **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. A predicate is a statement that may be true or false depending on the values of its variables.[1] It can be thought of as an operator or function that returns a value that is either true or false.
4. **A term:** In logic, the subject or predicate of a categorical proposition (q.v.), or statement.

**6a)**

1. **Design Strategy**: A system analyst or anyone who has the expertise in the niche is more proficient in the specification of the software. In order to accomplish the task of functional design, the software designing must be done by system designers. The system architect or designer makes use of primitive components to create a specification of a software artifact. Moreover, it involves both low-level component and algorithm design and high-level, architecture design.
2. **Programming Strategy:** Programming strategies must be made by experts. In addition, programmers must involve system designers to implement the software. This ensures that the functional and technical design goals are met. Software testing being an important element of the software quality assurance process requires you to perform it on a regular basis.
3. **Maintenance Strategy:** Maintenance strategy is a lifetime task and must be well written, as it might involve end-user for training. There are various kinds of software development models, with each one of them involving a strategy to perform a specific set of steps while the software is being developed.

**6b)** When a System or an application needs to be developed there will be an objective what that system is going to serve. Requirement document is written in order to make sure that the application is developed and tested in such a way that the application will serve the same objective once it’s released. In other words, all the expected functionalities out of the application are documented in terms of “Requirements” and this document is called a Requirement document. This Document is taken as a benchmark from various people in the project team like developers, testers, Business Analysts, etc. to understand the functional requirements of the application. It is also called an SRS document, which stands for System Requirement Specification Document.

**7a)**

1. **Preface:** This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
2. **Introduction:** This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
3. **Glossary:** This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
4. **User requirements definition:** The services provided for the user and the non-functional system requirements should be descried in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which must be followed should be specified.
5. **System architecture:** This chapter should present a high-level overview of the anticipated system architecture showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
6. **System requirements specification:** This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements, e.g, interfaces to other systems may be defined.
7. **System models:** This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
8. **System evolution:** This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
9. **Appendices:** These should provide detailed, specific information which is related to the application being developed. Examples of appendices that may be included are hardware and database descriptions. hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation of the data used by the system and the relationships between data.
10. **Index:** Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

**7b)**

1. Create an outline(Or use an SRS template)
2. Start with a purpose
3. Give an overview of what will be built
4. Add details to the specific requirements
5. Get approval for the SRS