1ai. Formal Methods

Model Formal

Checking Proofs

**Analytical Verification**

Abstraction Formal

Specification

**Formal Specification**

1aii. Formal methods can be applied at different stages of software development life cycle. On the basis of the details of the formal methods, some goals may be listed as follows:

1. Formal methods support in the creation of specifications that describe the true requirements of the user, which are not usually identical to the stated requirements. This can be achieved using formal methods because of the unambiguity of the formal specifications and the possibility to prove certain properties about it.
2. Formal methods ensure that the implementation of a particular software as well as hardware product should satisfy the requirements specification.
3. Formal methods are basically concerned for development and maintenance of security critical reliable systems on time and within budget. It increases trustworthiness of the system in the sense that the system developed is not just correct but known to be correct. Formal methods act as evidence which ensures that the system indeed satisfies the demand of security, reliability and correctness.

1b.

|  |  |
| --- | --- |
| Functional Requirements | Non-functional Requirements |
| 1. The end result of product feature. | The end result of product properties. |
| ii. It helps to verify the functionality of the software. | It helps to verify the performance of the software |
| iii. It focuses on user requirements. | It focuses on user expectation. |
| iv. Documentation describes what the product does. | Documentation describes how the product works. |

2a. Software engineering methodology:

* Waterfall
* Prototyping
* Incremental
* Spiral
* Rapid application development (RAD)
* Extreme Programming

### Waterfall development (a linear framework.): 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. The first formal description of the method is often cited as an article published by Winston W. Royce in 1970 although Royce did not use the term "waterfall" in this article.

The basic principles are:

* 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.

### Prototyping(an iterative framework): 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.

The basic principles are:

* Not a standalone, complete development methodology, but rather an approach to handling selected parts of a larger, more traditional development methodology (i.e. 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.
* User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation.
* Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users’ requirements.
* While most 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 problem.

Incremental development (a combined linear-iterative framework): 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.

The basic principles are:

* 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
* Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
* The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by iterative Prototyping, which culminates in installing the final prototype, a working system.

### Spiral development (a combined linear-iterative framework): 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.

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.

### Rapid application development (an iterative framework): Rapid application development (RAD) is a software development methodology, which involves iterative development and the construction of prototypes. Rapid application development is a term originally used to describe a software development process introduced by James Martin in 1991.

The basic principles 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 deadline.
* Generally includes joint application design (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.

2b. least rigorous spectrum rigor most rigorous

Occasional mathematical notation Fully formal specification

Embedded in English specification Language with a precise semantics

3ai. D**ifferences 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.

3aii. 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 |

3b. m1=mortal

m2=man

P1=every man is mortal

P2=smith is a man

[Smith is mortal]

∀(m2): m1 (p1, p2)

4a. o=object

m=monkey

~m=not monkey

∃(o):m v ~m

4b. Types of formal specification

* History-based specification
* State-based specification
* Transition-based specification

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](https://en.wikipedia.org/wiki/Z_notation" \o "Z notation), [VDM](https://en.wikipedia.org/wiki/Vienna_Development_Method" \o "Vienna Development Method) or [B](https://en.wikipedia.org/wiki/B-Method" \o "B-Method) 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[[3]](https://en.wikipedia.org/wiki/Formal_specification" \l "cite_note-lamsweerde-roadmap-3)

5ai.  A well-formed formula: In [mathematical logic](https://en.wikipedia.org/wiki/Mathematical_logic" \o "Mathematical logic), [propositional logic](https://en.wikipedia.org/wiki/Propositional_logic" \o "Propositional logic) and [predicate logic](https://en.wikipedia.org/wiki/Predicate_logic" \o "Predicate logic), a well-formed formula, abbreviated WFF or wff, often simply formula, is a finite [sequence](https://en.wikipedia.org/wiki/Sequence" \o "Sequence) of [symbols](https://en.wikipedia.org/wiki/Symbol_(formal)" \o "Symbol (formal)) from a given [alphabet](https://en.wikipedia.org/wiki/Alphabet_(computer_science)" \o "Alphabet (computer science)) that is part of a [formal language](https://en.wikipedia.org/wiki/Formal_language" \o "Formal language). A formal language can be identified with the set of formulas in the language. A formula is a [syntactic](https://en.wikipedia.org/wiki/Syntax_(logic)" \o "Syntax (logic)) object that can be given a semantic [meaning](https://en.wikipedia.org/wiki/Formal_semantics_(logic)" \o "Formal semantics (logic)) by means of an interpretation. Two key uses of formulas are in propositional logic and predicate logic.

aii. A quantifer: There are basically 2 types of quantifiers I.e universal and existential quantifers.

* Universal Quantifier: The expression: IMG_256x P(x), denotes the universal quantification of the atomic formula P(x). Translated into the English language, the expression is understood as: "For all x, P(x) holds", "for each x, P(x) holds" or "for every x, P(x) holds". IMG_257 is called the universal quantifier, and IMG_258x means all the objects x in the universe. If this is followed by P(x) then the meaning is that P(x) is true for every object x in the universe. For example, "All cars have wheels" could be transformed into the propositional form, IMG_259x P(x), where:
* P(x) is the predicate denoting: x has wheels, and
* the universe of discourse is only populated by cars.
* Existential Quantifier: The expression: IMG_256 xP(x), denotes the existential quantification of P(x). Translated into the English language, the expression could also be understood as: "There exists an x such that P(x)" or "There is at least one x such that P(x)" IMG_257 is called the existential quantifier, and IMG_258 x means at least one object x in the universe. If this is followed by P(x) then the meaning is that P(x) is true for at least one object x of the universe. For example, "Someone loves you" could be transformed into the propositional form, IMG_259 x P(x), where:
* P(x) is the predicate meaning: x loves you,
* The universe of discourse contains (but is not limited to) all living creatures.

aiii. A predicate: A predicate is a statement that contains variables (predicate variables), and they may be true or false depending on the values of these variables.

aiv. A term: **Term**, in [logic](https://www.britannica.com/topic/logic), the subject or [predicate](https://www.merriam-webster.com/dictionary/predicate) of a [categorical proposition](https://www.britannica.com/topic/categorical-proposition) (q.v.), or statement.

6a. Software development strategies:

* Specification or functional design, done by system analysts in consort with the potential end users of the software to determine why to do this, what the application will do, and for whom it will do it.
* Architecture or technical design, done by system designers as the way to achieve the goals of the functional design using the computer systems available, or to be acquired, in the context of the enterprise as it now operates. This is how the system will function.
* Programming or implementation, done by computer programmers together with the system designers.

6b. A product requirements document (PRD) is a [document](https://en.wikipedia.org/wiki/Document" \o "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](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)]

PRDs are most frequently written for [software](https://en.wikipedia.org/wiki/Software" \o "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](https://en.wikipedia.org/wiki/Marketing_Requirements_Document" \o "Marketing Requirements Document) (MRD)). The requirements are then [analyzed](https://en.wikipedia.org/wiki/Requirements_analysis" \o "Requirements analysis) by a (potential) maker/supplier from a more technical point of view, broken down and detailed in a [Functional Specification](https://en.wikipedia.org/wiki/Functional_Specification" \o "Functional Specification) (sometimes also called Technical Requirements Document).

7a. Structure of a requirement document:

* Title & author information
* Purpose and [scope](https://en.wikipedia.org/wiki/Scope_(project_management)" \o "Scope (project management)), from both a technical and business perspective
* [Stakeholder](https://en.wikipedia.org/wiki/Project_stakeholder" \o "Project stakeholder) identification
* Market assessment and target [demographics](https://en.wikipedia.org/wiki/Demographics" \o "Demographics)
* Product overview and [use cases](https://en.wikipedia.org/wiki/Use_case" \o "Use case)
* [Requirements](https://en.wikipedia.org/wiki/Requirement" \o "Requirement), including
  + [functional requirements](https://en.wikipedia.org/wiki/Functional_requirement" \o "Functional requirement) (e.g. what a product should do)
  + [usability](https://en.wikipedia.org/wiki/Usability" \o "Usability) requirements
  + technical requirements (e.g. security, network, platform, integration, client)
  + environmental requirements
  + support requirements
  + interaction requirements (e.g. how the product should work with other systems)
* Assumptions
* Constraints
* Dependencies
* High level workflow plans, timelines and milestones (more detail is defined through a [project plan](https://en.wikipedia.org/wiki/Project_plan" \o "Project plan))
* Evaluation plan and performance [metrics](https://en.wikipedia.org/wiki/Metric_(mathematics)" \o "Metric (mathematics))

7b. Steps involved in writing a system requirement specification

1. Make an outline.
2. Define the purpose of your product.
3. Describe what you're building.
4. Detail the requirements.
5. Get it approved.