NAME: IGE .A. SAMUEL

MATRIC N0: 17/SCI01/039

COURSE: CSC 308(Formal Methods and Software Development) ASSIGNMENT.

1a.)

1aii.) The major reason for considering formal method is because they consist of a formal specification language and employ a collection of tools to support the syntax checking of the specification, as well as the proof of properties of the specification. They allow questions to be asked about what the system does independently of the implementation. The use of mathematical notation avoids speculation about the meaning of phrases in an imprecisely worded natural language description of a system.

1c.) -Formal methods are intended to systematize and introduce rigor into all the phases of software development. This helps us to avoid overlooking critical issues, provides a standard means to record various assumptions and decisions, and forms a basis for consistency among many related activities. WHILE

 -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.) i. **Rapid-application development** (**RAD**), also called **rapid-application building** (**RAB**), is both a general term, used to refer to adaptive [software development](https://en.wikipedia.org/wiki/Software_development) approaches, as well as the name for [James Martin](https://en.wikipedia.org/wiki/James_Martin_%28author%29)'s approach to rapid development. In general, RAD approaches to software development put less emphasis on planning and more emphasis on an adaptive process. [Prototypes](https://en.wikipedia.org/wiki/Software_prototype) are often used in addition to or sometimes even in place of design specifications.

ii. The **waterfall model**: is a breakdown of project activities into linear [sequential](https://en.wikipedia.org/wiki/Sequence) phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks. The approach is typical for certain areas of [engineering design](https://en.wikipedia.org/wiki/Engineering_design). In [software development](https://en.wikipedia.org/wiki/Software_development_process), it tends to be among the less iterative and flexible approaches, as progress flows in largely one direction ("downwards" like a [waterfall](https://en.wikipedia.org/wiki/Waterfall)) through the phases of conception, initiation, [analysis](https://en.wikipedia.org/wiki/Analysis), [design](https://en.wikipedia.org/wiki/Software_design), [construction](https://en.wikipedia.org/wiki/Software_construction), [testing](https://en.wikipedia.org/wiki/Software_testing), [deployment](https://en.wikipedia.org/wiki/Implementation) and [maintenance](https://en.wikipedia.org/wiki/Software_maintenance).

iii. **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).

iv. **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 enhance s collaboration between the departments responsible for different segments of the developmentlife cycle, such as development, quality assurance, and operations.

2b.) They are techniques and tools based on mathematical and formal logic and can achieve various forms of vigour.

3ai.) **Propositional logic** studies the ways statements can interact with each other (A,B,C). It is important to remember that propositional logic does not really care about the content of the statements. WHILE

Predicate logic is usually used as a synonym for first-order logic, but sometimes it is used to refer to other logics that have similar syntax.

3aii)



3b)

P1=mortal

P2=man

X1=every man is mortal

X2=smith is a man

Smith is mortal

(∀)(P2) P1 (X1, p2) OR

∀(man(x) mortal(x)

4a. o=object

m=monkey

~m=not monkey

∃(o):m v ~m

4b

1. **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

1. **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.

ii. **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.

5a.)i. **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.ii. **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 existential quantifier are 'some,' 'at least one,' and 'there.iii. **A predicate**: state, affirm, or assert (something) about the subject of a sentence or an argument of a proposition

6ai. **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 theuse 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.

1. **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!

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)

7b1. Introduction1.1 Purpose1.2 Intended Audience1.3 Intended Use1.4 Scope1.5 Definitions and Acronyms2. Overall Description2.1 User Needs2.2 Assumptions and Dependencies3. System Features and Requirements3.1 Functional Requirements3.2 External Interface Requirements3.3 System Features3.4 Non functional RequirementsOnce 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? Youshould 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
* SoftwareCommunications
* 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:PerformanceSafetySecurityQualityThe 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 SRSOnce 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.