IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 3, Ver. III (May - June 2017), PP 08-16
www.iosrjournals.org

Software Quality Specification Using Markov Chain Statistical Technique

D. G. Iyanda[1*], Ogundele, L. A.[2]and Mojeed, T.A[3].
[1]*Department of International Co-operation, State Planning Commission of Osun, New Secretariat, P. M. B. 4435, Osogbo, Osun State, Nigeria.*
[2]*Department of Computer Science, Osun State College of Education, Ilesa, Osun State, Nigeria.*
[3]*Department of Research and Marketing, Punch Nigeria Limited, Lagos, Nigeria*

Abstract: *Informal software engineering methods have been using combination of diagrams, text, tables and simple notation to create analysis and design models with application of little mathematical rigour. On the other hand, formal methods allow a software engineer to create a model that is more complete, consistent, and unambiguous than those produced using conventional methods (Roger, 2005). However, it had been observed that the fundamental step in the Markov analysis of a software specification is to define the underlying probability law for the usage of the software under consideration. From an analytical point of view, this is a traceable stochastic process and a good basis for statistical testing. Also, from the software engineering point of view, definitions and properties of the model ensure the testability of the software. All the available models have their drawbacks and none of them used mathematical method, hence our Markov chain statistical approach. In this approach, the analysis of the specification, performed prior to design and coding, yields an irreducible Markov chain called Usage Markov Chain Statistics. Each usage state is labelled with a stimulus from the input domain of the software. If the control is exerted depend both on the position at time t, and position at time t-1, then , there is succession of state such that the dependency of the past state reaches a fixed distance into the past, for instance, if it reaches back three states, then it forms a Markov Chain statistical approach.*

Keywords: *Markov Chain, Software Development, Software Engineering, Software Evolution, Software Quality, Software Specification, Software Validation*

I. Introduction

As stated in Dijkstra (2004), the term Software Engineering was coined in 1968 by F.L. Bauer of the Technology University of Munich,it was welcomed and then interpreted as an apt reflection of the fact that the design of software system was an activity per excellence for the mathematical engineer. In the mean time, software engineering has become an almost empty term, as demonstrated by Data General, who overnight promoted all its programmers to the exalted rank of “Software Engineer”. Software engineering is defined as the application of systematic, disciplined, quantifiable approach to the operation, maintenance and development of software (Soriyan, 2004). It traditionally focuses on the practical means of developing software (Warsta, 2002).IEEE (1990) defines software engineering as application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. In addition, it draws on knowledge from fields such as computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics and systems engineering. Robert *et al* (2006) define it to reflect the viewpoint of the software project management and stated that it is a disciplined, systematic approach to the development, operation, maintenance, and retirement of software through the practical application of scientific knowledge and processes.

Bauer (1968) stated that, Software engineering encompasses knowledge, tools, and methods for defining software requirements, perform software design, software construction, software testing and software maintenance. In Fadi *et al* (2005), software engineering is described as a cognitive reaction to the complexity of software development. It was also added that, it reflects the inevitable need for analysis and planning; reliability and control of risk; and scheduling and coordination when embarking on any complex human endeavour. Developing a successful software solution involves establishing a destination or product goal; carefully examining alternative designs for the system; identifying potential risks, demarcating milestones along the way; identifying what activities must be done in what sequence or those that may be done in parallel; and identifying needed resources-including human resources, financial resources, information, tools and strategies.

Roger (2005) defined software engineering as, the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. He then added that the definition failed to mention technical aspects of software quality and to address the need for customer satisfaction or timely product delivery. The author submitted that the definition do omitted the importance of measurement and metrics but provided us with a baseline. He then asked for sound engineering principles, economically built software so that it is reliable and work efficiently on many different “real machines”. He also said that we need adaptability and agility, that is, dexterity, quickness and nimbleness which encompass quality, skilfulness in the use and rate of performance.The purpose of software engineering is to prepare reused software (whether from Cleanroom environments or not) for incorporation into the software product. The functional semantics and interface syntax of the reused software must be understood and documented, and if incomplete, can be recovered through function abstraction and correctness verification. Also, the certification goals for the project must be achieved by determining the fitness for use of the reused software through usage models and statistical testing. The wide use of software systems has made software engineering an integral part of human existence (Soriyan, 2004).

II. Software development process

Software engineering is an engineering discipline that concerns the development of software. As an engineering discipline it is focused on making things work, applying theories, methods, and tools when and where they are appropriate. Today, software systems are defined as a set of instructions designed to perform specific task. They are usually developed to automate existing manual systems in order to eradicate the barriers of distance, time and to ensure quality performance of systems.Software development process is a transformation process executed by series of agents (people and machine) which perform a variety of activities and whose association results in the production of a software system (Soriyan, 2004). It also refers to the sequence of steps that is required to develop or maintain software. Salo (2006) emphasizes that software development process is aimed at providing the technological framework for applying tools and people to the software task. The essence of systems development process include ensuring that high quality systems are delivered, providing strong management controls over the projects, and maximizing the productivity of the development team (Bender, 2003). In today’s software development process, quality requirements during architectural design and decision are fundamental issues to the stakeholders (developers, users etc.). The challenge in software development is to develop software with the right quality levels. It is, however, unfortunate that the right quality of existing software systems is not met. As a result, the quality of the system must be taken into consideration during software development.

Software development is usually performed in a development project. The project identifies requirements from a customer or intended market for the system. Then it analyses, designs, and implements a software system that will fulfil these requirements. Several tasks are performed by the people in the development project resulting in the creation of a number of artefacts’, e.g., requirements specifications, prototypes, and an implementation. In order to make the tasks easier and more structured, the people in the development project usually follow a software development process.A software development process describes activities that have to be done and when, during the development, they should be done. A process can also describe which roles that should exist in the development organisation. Examples of roles are requirements engineer, programmer, software architect, tester, and manager. The process defines each role's tasks and responsibilities. Depending on how far the development has progressed, the number of people in each role may vary. In essence, the process helps to structure the developers’ work and make the development process less unpredictable.

There are different processes used to develop software systems. These processes are characterized by fundamental activities or phases in accordance with the method or model. In the case of the waterfall model, the following activities are involved:

i. Project Establishment: This phase initiates the software development process. It can be defined as a systematic technique that supports the clarification and negotiation of the aim, level of ambition, scope and conditions of the software development project (Kensing *et al.*, 1996). In Lanzara’s terms, project establishment is a reframing process (Lanzara, 1982). It suggests the activities the software development team will carry out in the software development process, which tools and techniques will be appropriate for the software development and a sound basis for the succeeding project activities. It involves the identification of critical success factors such as the objectives of the software, meetings where the conditions of the projects are negotiated by the software development team, project planning and negotiation, project charter which is the basis for the development team and the steering committee decisions on how to develop the software.
ii. Requirement Analysis: The purpose of the requirement analysis phase is to ensure that the requirements and specifications of the software are feasible, consistent and well understood by the software development team. This phase involves the gathering of the requirements from the user community usually via fact-finding techniques such as questionnaires, interviews and focus groups. The key activities involved include resolving ambiguities and discrepancies in the requirement documents, analysing and classifying

requirement, identifying requirements and specifications that are missing, conflicting, ambiguous and infeasible, and the use of object-oriented analysis to verify the specifications (Bender, 2003). This phase is critical to the success of the software development process. Whitten *et al*. (2001) stated that the identification of requirements may seem trivial but it is often the source of many errors, omissions and conflict. It is, therefore, a phase that must not be omitted in the software development process. The deliverable of this phase is a business requirement statement.

iii. Design Phase: This phase is a highly significant phase in the software development process where the software designer plans how a software system should be developed in order to make it functional, reliable, and reasonably easy to understand, modify and maintain (Aggarwal *et al.,* 2005). The purpose of the design phase is to transform the business requirement statement from the requirement analysis phase into a design specification. In other words, the design phase addresses how technology will be used in the system. It also requires adherence to internal and external design standards that ensure completeness, usability, performance and quality (Bender, 2003). The design phase is usually a two-part iterative process which includes the conceptual design and the technical design (Aggarwal *et al.,* 2005). The conceptual design describes the functions of the system while the technical design describes the hardware configuration, the software needs, the communication interfaces, the input and output of the system and the network configuration.

iv. Coding and unit testing: The coding phase involves the reduction of the software design to codes. Unit testing is the process of taking a sub-system and running it in isolation from the rest of the software products by using prepared test cases and comparing it with the actual result predicted by the specifications and design of the sub-system (Aggarwal *et al.,* 2005). The purposes of unit testing include:

a) Easy location of errors when compared to testing the whole system; and

b) Elimination of confusing interactions of multiple errors in different parts of the software.

v. System Integration: This phase follows the coding and unit testing phase of the software development process. It can be defined as the combination of diverse application entities into a relationship which functions as a whole, i.e., the joint collaboration of several system components to form a single system (Aguilar, 2005). It can also be defined as the successful putting together of various components, assemblies, and subsystems of a system and having them work together to perform what the system was intended to do. During this phase of the software development process, independently developed subsystems which may be developed by different sub-developers from different organisations are assembled. System integration is one of the most costly and time-consuming activity in the software development process. For large and complex systems, up to 40% of the development time may be used in this activity (Somerville, 1998). There are two basic approaches used in the system integration phase. These approaches include the “big bang” approach and the incremental approach (Somerville, 1998). In the “big bang” approach, all the sub-systems are assembled at the same time. This approach is usually unsuccessful because it increases the cost of error location. Incremental integration involves putting the sub-systems together in testable increments. The advantage of this approach over the “big bang” approach is that as problems emerge in the integration process, it is likely that they result from the integration of a new sub-system. There are a lot of problems associated with system integration. They include:

a) Management Problems: These are problems that result from the developers or sub-developers model (Somerville, 1998). Management problems usually arise when the sub-systems are not available for integration when required.

b) Interfacing Problems: This usually arises from misunderstanding from different organisations developing different parts of the software systems. Interfacing problems can be classified as physical interfacing, incomplete/incompatible interfacing and interface misunderstanding. Physical interfacing problem arises when systems cannot be connected as a result of the incompatibilities that exist between them. A typical example is software systems that are developed on different system platforms and different programming languages. Incomplete/ incompatible interfacing problem usually arises when sub-systems provide interfaces which do not allow information to be exchanged among them. Interface misinterpretation occurs when different organisations interpret interface specifications in different ways.

c) Configuration problems: This arises when the wrong version of sub-systems and components are included in the systems to be integrated.

The iterative method is another development process that prescribes the construction of initially small but even larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because they allow a potential of reaching the design goals of a customer who does not know how to define what he/she wants. Today, the agile software development processes are built on the foundation of iterative development. Agile software development is an umbrella term for a collection of development methodologies that focus on adaptability over predictability. To that foundation they add a lighter, more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software. With agile method of development, projects tend to be separated into smaller, more manageable phases that can be quickly completed (e.g. within one to four weeks) and then either released into production or expanded upon with additional initiatives in a rolling fashion. Agile practices encourage frequent direct communication to supplement written documentation, and success is ultimately dictated by software that works.

Consequently, research in software engineering and development has incorporated software architecture as a phase in the development cycle in between the requirement and design phases. Also software architecture is a discipline pervading all phases of development. During the requirements, architecture may be used to identify, prioritize, and record system concerns and desired. During design and analysis, architecture may be used to model, visualise, and analyse design decisions chosen to address the principal concerns and achieve the desired qualities. Decisions may be guided by adopting one or more architectural styles. During implementation and testing, architecture may be used to drive testing, instantiate a product, support runtime dynamism, or enforce security policies. Rather than throw out architecture at this point, as is often done, architecture remains part of the product. During maintenance, architecture may be used as a basis for incorporating new features, or increasing modelling detail.Most software development processes can be described using a general set of activities (or collections of activities) for software development. These have been described by (Sommerville, 2001) as:Software Specification, Software Development, Software Validation and Software Evolution

Software Specification: This activity concerns the identification of requirements from the intended users of the system. The requirements are verified and prioritised so that the most important requirements can be identified. Having a clear set of requirements is important for the success of the project. If the wrong functionality is implemented it does not matter how well it is implemented, it is still wrong.

Software Development: The requirements are used as input for the creation or modification of software architecture for the system. The architecture describes the high level components of the system, their responsibilities, and interactions. The architecture is then further refined into a more detailed design for the system. Finally, the implementation is done according to the design.

Software Validation: The system has to be validated to make sure that it fulfils the requirements gathered during the specification phase. Defects that are identified should be corrected so that the software system that is delivered to the customer is as correct as possible. Many methods for testing a system exist and the method used depends on the requirement that is being tested.

Software Evolution: The development of a software system does not end once it has been delivered to the customer. A system might be in use for ten to twenty years and the likelihood that changes have to be made to it during this time is high. New functionality can be added and changes to existing functionality can occur. These basic activities are present in most software development processes, for example, waterfall (Pfleeger, 1998) and iterative development methods (Larman *et al.*, 2003) such as Extreme Programming (Beck, 2000). In the waterfall development method, the development progresses from one activity to the following, putting the focus on one activity at a time. In iterative development, the activities of the development process are repeated over a number of smaller iterations. Some requirements may be changed in each iteration so that the software system grows incrementally and adapts to changing requirements. In the model described by (Sommerville, 2001), software architecture is found in the second step, i.e., software development. The software architecture is used as input to the continued design of the software system. Sommerville continues to describe a set of design activities within the software development activity:

i. Architectural Design. The activity of identifying the sub-systems of the software system. The sub-systems' relations are identified and documented.
ii. Abstract Specification. Each sub-system that was identified is assigned responsibilities and constraints that it has to fulfill. The necessary services are identified and documented.
iii. Interface Design. The interfaces that enable the interaction between the sub-systems are designed.
iv. Component Design. The responsibility for the services previously identified is assigned to components. The interfaces for the components are designed.
v. Data structure Design. The data structures that will be used in the services and for communication between services are specified and documented.
vi. Algorithm Design. The algorithms that will be used in the services are selected or designed.

Software Quality

To understand the landscape of software quality it is central to answer the so often asked question: *what isquality*? As this concept becomes clear, then it is easier to understand the different structures of quality available at the market. Many prominent authors and researchers have provided an answer to thatquestion. This work is not meant to introduce yet another answer to what quality is, but intends to answer the question by studying the answers that some of the more prominent gurus of the quality management community have provided. Basically, there are two major camps whendiscussing the meaning and definition of (software) quality (Hoyer *et al.,* 2001): In this work we identify that there are two major camps when discussing the meaning and definition of (software) quality (Hoyer *et al.,* 2001):

a) Conformance to specification: Quality is defined as a matter of products and services whose measurablecharacteristics satisfy a fixed specification.

b) Meeting customer needs: Quality is identified independent of any measurable characteristics. That is,quality is defined as the products or services capability to meet customer expectations.
Consequently, quality is not an exact concept because everyone looks at quality from their ownviewpoint. Evans and Lindsay define quality from five perspectives in Evans *et al.,* (1999):

i. Quality is judgmental criteria. This defines quality as the goodness of aproduct. This definition is referred to as a prevalent definition of quality.The definition means that quality is absolute and universal. This kind ofquality cannot be defined exactly; you just know it when you see it. Thiskind of definition has little practical value for managers; it does not providea way to measure or assess quality.

ii. Quality is product-based criteria. This defines quality as a function of aspecific, measurable variable that differs in quantity with some productattributes.

iii. Quality is user-based criteria. This defines fitness for intended use, or howwell the product performs its intended function. This quality definition isbased on the presumption that quality is determined by what a customerwants.

iv. Quality is value-based criteria. As the name says, it is based on value. Thatis, the relationship between usefulness and satisfaction with price. Thismeans that the product is of good quality when it offers the sameusefulness as the competitors' product, but at a lower price, or it offersbetter usefulness at the same price.

v. Quality is manufacturing-based criteria. This defines quality as a desirableoutcome ofdesign and manufacturing practice or conformance to thespecification.

c) The International Organisation for Standardization (ISO) defines quality in theISO 9000 definition (Praxiom Research Group (2006)) as a characteristic that aproduct or service must have. For example, a product must be reliable and aservice must be efficient. However, not all qualities are equal, some are moreimportant than others. The qualities the user wants are of the most importance;these are qualities the product and service must have. Therefore, the quality of aproduct or service is the one that meets the wants and expectations of the user. From the listed points one can deduce that the factors to determine the quality of aproduct or service as stated by Perry and Wolf (1992) are: (i) Correctness, C (ii) Reliability, R (iii) Efficiency, E (iv) Integrity, I (v) Usability, U(vi) Maintainability, M (vii) Testability, T (viii) Flexibility, F (ix) Portability, P(x) Reusability Ru and(xi) Interoperability, In.

III. Related Works

It has been observed that there are many existing software engineering specification models with their attendant drawbacks. Some of the models are as discussed below.Nabil and Govardhan (2010), observed that the advantages of Waterfall Model for software specification include; easy to understand and implement, widely used and known (in theory),reinforces good habits: define-before- design,design-before-code, identifies deliverables and milestones, document driven, URD, SRD, … etc. publisheddocumentation standards, e.g. PSS-05, and works well on mature products and weak teams. However, the weakness of this model according to Karlm (2006), is that its administrative overhead, is significant costly for small teams and projects, Other disadvantages are: idealized, doesn't match reality well, the model doesn't reflect iterative nature of exploratory development, unrealistic to expect accurate requirements so early in project, software is delivered late in project, delays discovery of serious errors, difficult to integrate risk management and expensive to make changes to documents, "swimming upstream. The problems with the Waterfall Model created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility called Iterative Development Model, Nabil and Govardhan (2010). With Iterative Development Model, the project is divided into small parts. This therefore allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products, which are produced at the end of each step (or series of steps), can go into production immediately as incremental releases. The V-Shaped Model also has the following merit: simple and easy to use, each phase has specific deliverables higher chance of success over the waterfall model due to the early development of test plans during the life cycle, works well for small projects where requirements are easily understood. This model when carefully studied has the following drawbacks: very rigid like the waterfall model, little flexibility and adjusting scope is difficult and expensive, software is

developed during the implementation phase, so no early prototypes of the software are produced, the model does not provide a clear path for problems found during testing phases, Rlewallen (2005).
Another model is the Spiral Model which is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. The merits of this model include high amount of risk analysis, good for large and mission-critical projects and software is produced early in the software life cycle. One of the demerits of this model is that it costly to use. Also, risk analysis of the model requires highly specific expertise; project's success is highly dependent on the risk analysis phase and doesn't work well for smaller projects, Rlewallen (2005).
Extreme Programming Model is a model with the following advantages: Lightweight methods suit small-medium size projects, it produces good team cohesion, emphasises final product, iterative and test based approach to requirements and quality assurance. The weaknesses of this model according to Karlm (2006) are: difficult to scale up to large projects where documentation is essential, needs experience and skill if not to degenerate into code-and-fix programming pairs is costly, test case construction is a difficult and specialized skill.

Proposed Markov Chain Statistical Model

Markov Chain proves to be a good model for software engineering for several reasons. From the software engineering point of view, definitions and properties of the model ensure the testability of the software. Once the model has been built, any number of statistically typical test cases can be obtained from the model. From an analytical point of view, this is a traceable stochastic process and a good basis for statistical testing. Unlike all other models earlier discussed, the fundamental step in the Markov analysis of a software specification is to define the underlying probability law for the usage of the software under consideration. The analysis of the specification, performed prior to design and coding, yields an irreducible Markov chain called usage Markov Chain. This claim has unique start state $\mathbf{S_o}$(which represents invocation of the software), a unique final state $\mathbf{S_F}$ (which represents termination of the software) and a set of intermediate usage states, $\mathbf{S_i}$. Each usage state is labelled with a stimulus from the input domain of the software. The state set,$\{S = \{S_0, \{Si\}, S_F\}\}$ is ordered by the probabilistic transition relation,$\{SX(0,1)XS\}$. For each arc defined by this relation, the next state is independent of all past states given the present state. This is called the Markov property. This method of creating states yields a directed graph whose states are labelled with entries drawn from the input domain of the software. Noting that the factors to determine the quality of aproduct or service as stated by Perry and Wolf (1992) are: (i) Correctness, C (ii) Reliability, R (iii) Efficiency, E (iv) Integrity, I (v) Usability, U (vi) Maintainability, M (vii) Testability, T (viii) Flexibility, F (ix) Portability, P (x) Reusability Ru and (xi) Interoperability, In. The arcs of the graph therefore define an ordering that the input must be obeyed. This is shown in the Figure 1 below.

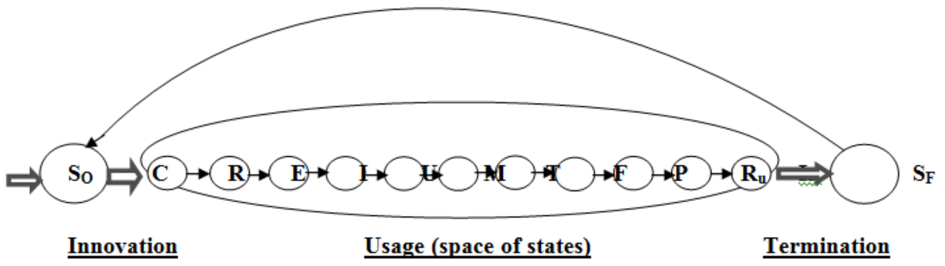

Figure 1: Markov Chain Diagram

From the above, it implies that to formulate a probabilistic device as a Markov chain, the space of states should be selected so that each state contains all relevant information about the past. If the control is exerted depend both on the position at time *t*, and position at time *t-1*, then
$P\{X_{t+1} = k|X_t=j, X_{t-1}= i \text{ ....}\}$ depends on both j and i and we get the Markov property by taking successive pairs of positions (i,j) for the states, then
$P(k|j, i) =P\{X_{t+1} = k|X_t=j, X_{t-1}= i\}$, it follows that
$PX\{_{t+2}=l,X_{t+1}=k|X_t=j,X_{t-1}=i,..........\}$
$= P\ X\{_{t+2}=l|X_{t+1}=k,X_t=j,X_{t+1}= i,..........\ .\ P\ \ X_{t+1}=k|X_t=j,X\}_{t-1}\{=i,...........\}$
$= Pl|\{k,jP\ k\ |\}j,i.\ (\ )$
Thus, the pair (X_t, X_{t+1}) here form a Markov Chain with transition probabilities

P{(l,k)/(j,i)}= P(l|k,j)p(k|j,i). Generally if there is succession of state such that the dependency of the past state reaches a fixed distance into the past, for instance, it reaches back three states, then; $P(X_{t+1}=i_{t+1}|X_t=i_t,X_{t-1}=i_{t-1},.........) = P(X_{t+1}= i_{t+1}|X_t=i_t, X_{t-1}=i_{t-1},X_{t-2}=i_{t-2})$ in this case the triple (X_t,X_{t-1}, X_{t-2}) form a Markov chain. The application of this case to the proposed model gives $(X_t,X_{t-1}, X_{t-2},X_{t-3}, X_{t-4},X_{t-5}X_{t-6}, X_{t-7},X_{t-8}, X_{t-9}, X_{t-10})$ where X represents each factor being considered and the subscripts t, t-1, t-2, ...,t-10 represent the time at the respective location of the factors being considered.

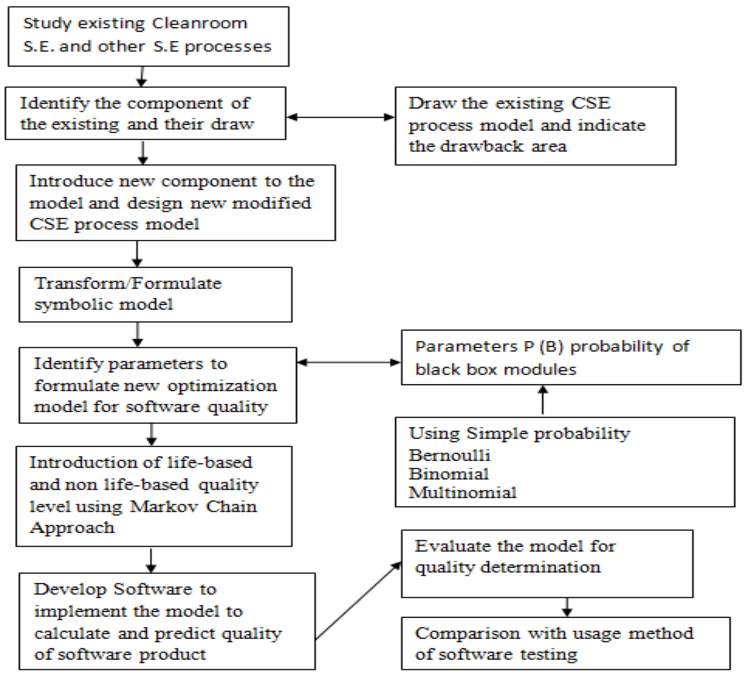
Study existing Cleanroom S.E. and other S.E processes
Identify the component of the existing and their draw
Draw the existing CSE process model and indicate the drawback area
Introduce new component to the model and design new modified CSE process model
Transform/Formulate symbolic model
Identify parameters to formulate new optimization model for software quality
Parameters P (B) probability of black box modules
Using Simple probability Bernoulli Binomial Multinomial
Introduction of life-based and non life-based quality level using Markov Chain Approach
Evaluate the model for quality determination
Develop Software to implement the model to calculate and predict quality of software product
Comparison with usage method of software testing

Figure 2: Framework of the Proposed Markov Chain Statistical Model

The Statistical feature of the Model

The statistical characteristic of the model is Binomial Distribution. Binomial Distribution is a discrete probability distribution based on Bernoulli process. In experiment there are only two mutually exclusive and collectively exhaustive events. The probability of occurrence of the events of the experiments is the same in all the trials. In all the "*n"* trials the observation are independent of one another. Based on these fundamentals, the general Binomial distribution is represented as follows:

$$(p+q)^n = \sum_{x=0}^{n} \binom{n}{x} p^x q^{n-x} \qquad \text{..............................Equation 1}$$

$$p = probabilityofsuccess,$$

$$q = probabilityoffailure,$$

$$x = 0,1,2,3,4,5,\ldots..n = No.oftrials$$

As earlier described, the parameters to be considered are: (i) Correctness, C (ii) Reliability, R (iii) Efficiency E (iv) Integrity, I (v) Usability, U (vi) Maintainability, M (vii) Testability (viii) Flexibility (ix) Portability, P (x) Reusability, (xi) Interoperability, I. The model will consider the parameter on the assumption that 0, 1, 2, 3, 4, ...11 are present in given software to test its quality. Obviously, if any of the parameter is missing, the probability value is less than 1. This implies that the quality of the software has been compromised. For the purpose of ascertaining the quality of the software, all the parameter must be available. Here, the probability of success is "*x"* which is 0.5 and probability of failure is "*a"* which is also 0.5 while the total number of parameters being considered is 11. Following from Binomial Distribution formula, the probability of presence of each parameter could be computed as follows:

For 0 – parameter $: = \binom{11}{0} x^0 a^{11}$ ....equation 2

For 1 – parameter $: = \binom{11}{1} x^1 a^{10}$ .... equation 3

For 2 – parameters $:= \binom{11}{2}x^2a^9$ .... equation 4

For 3 – parameters $:= \binom{11}{3}x^3a^8$ .... equation 5

For 4 – parameters $:= \binom{11}{4}x^4a^7$ .... equation 6

For 5 – parameters $:= \binom{11}{5}x^5a^6$ .... equation 7

For 6 – parameters $:= \binom{11}{6}x^6a^5$ .... equation 8

For 7 – parameters $:= \binom{11}{7}x^7a^4$ .... equation 9

For 8 – parameters $:= \binom{11}{8}x^8a^3$ .... equation 10

For 9 – parameters $:= \binom{11}{9}x^9a^2$ equation11

For 10 – parameters $:= \binom{11}{10}x^{10}a^1$ .... equation 12

For 11 – parameters $:= \binom{11}{11}x^{11}a^0$ .... equation 13

The quality of the software could therefore be determined by considering the total probabilities of all parameters available. This could be easily computed using the adopted Binomial equation below.

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k} \quad \text{..equation 14}$$

where;

$x = probabilityof\ factors\ present,$

$a = probabilityoffactorsnotpresent,$

$k = 0,1,2,3,4,5, \ldots..11 = Noofparametersconsidered$

In Binomial Distribution Method (BDM), there are only two mutually exclusive and collectively exclusive events, therefore probability of occurrence of the events of the experiments is the same in all trial. In all the n trials, the observations are independent of one another. Based on this fundamentals, the binomial probability distribution could represented as

$P(X \text{ successes in n trials given } p) = {}^{n}C_{X}p^{x}q^{n-x}$ where X =0,1,2,3,…..,n

The above distribution could be further represented as $P(X, n, p)$, in ***n*** trials given; $p = {}^{n}C_{X}p^{x}q^{n-x}$ where X = 0,1,2,3,…..,n

where ***n*** is the number of trials; ***p*** is the probability of factors present in a trial; ***q*** is the probability of factor absent in a trial ***(1-p),X*** is the random variable representing the number of factors present and ***P(X ,n,p)*** the probability of having ***X*** factors in ***n*** trials if the probability of factor present in a trial is ***p***. Hence, the cumulative distribution function of the binomial distribution is:

$$F(X,n,p) = \sum_{X=0}^{n} p(X,n,p) = \sum_{X=0}^{n} \left({}^{n}C_{X}p^{x}q^{n-x}\right) = 1 \quad \text{..................equation 15}$$

IV. Conclusion

It had been observed that the fundamental step in the Markov analysis of a software specification is to define the underlying probability law for the usage of the software under consideration. From an analytical point of view, this is a traceable stochastic process and a good basis for statistical testing. All the available models have their drawbacks and none of them used statistical method, hence our Markov chain statistical approach. The statistical characteristic of the model is Binomial Distribution. Binomial Distribution is a discrete probability distribution based on Bernoulli process. In experiment there are only two mutually exclusive and collectively exhaustive events. The probability of occurrence of the events of the experiments is the same in all the trials. In all the "*n"* trials the observation are independent of one another. Based on these fundamentals, the general Binomial distribution is useful for the achievement of the expected result. Further work on the implementation of the model is in progress.

References

[1] Aggarwal, K., and Yogis, S. (2005). Software Engineering: Programs, Documentation and Operating Procedures. New Age International Publishers, New Delhi

[2] Aguilar, A. (2005). Semantic Interoperability in the Context of e-health. Digital

[3] Enterprise Research Institute, National University of Ireland, Galway, Ireland.

[4] Bauer, F. L. (1968). Software Engineering, NATO Software Engineering Conference, Garmisch, Germany. Edited by P. Naur and B. Randell, published January 1969. Software Engineering: Report of a conference. www.Wikipedia.org

[5] Beck, K. (2000): Extreme Programming Explained. ISBN: 0-201-61641-6, Addison-Wesley.

[6] Bender RBT Inc. (2003). Systems Development Lifecycle: Objectives and Requirements.Bender RBT Inc, Queensbury, New York.

[7] Dijkstra E W. (2004). There is still a war going on manuscript, E.W. Dijkstra Archive. The University of Texas, Austin.

[8] Fadi P. Deek, James A. M. McHugh and Osama M. Eljabiri (2005). Strategic Software

[9] Engineering An Interdisciplinary Approach, Auerbach Publications, USA.

[10] Hoyer, R., and Hoyer, B. (2001). "What is quality?", Quality Progress, no. 7, pp. 52-62.

[11] IEEE (1990). Standard Glossary of Software Engineering Terminology, IEEE Std 610.12- 1990. www.wkipwdia.org.

[12] Karlm (2006), "Software Lifecycle Models', KTH,2006 .

[13] Lanzara, G. (1982), The Design Process: Frames, Metaphors and Games: System Design for, with, and by the Users. In Proceedings of Participatory Design Conference Design Conference, Cambridge Massachusetts U.S.A.

[14] Larman, C., Basili, V. R. (2003): "Iterative and Incremental Developments.A Brief History," Computer, vol. 36, pp. 47-56.

[15] Nabil Mohammed Ali Munassar and A. Govardhan (2010): A Comparison Between Five Models Of Software Engineering; International Journal of Computer Science Issues, Vol. 7, Issue 5,

[16] Pfleeger, S. L (1998): Software Engineering: Theory and Practise,intl. edition. ISBN: 0-13-081272-2, Prentice Hall.

[17] Rlewallen (2005): "Software Development Life Cycle Models" http://codebeter.com.

[18] Robert T. F., Donald F. S. and Linda I. S. (2006). Quality Software ProjectManagement, Published by Pearson Education Inc, London.

[19] Roger S. P. (2005). Software Engineering A Practitioner’s Approach, 6th Edition, Mc Graw Hill, New York. pp. 796-815.

[20] Salo, O. (2006). Enabling Software Process Improvement in Agile Software Development Teams and Organizations. VTT Publications.

[21] Sommerville, I. (1998). Software Intensive Systems Engineering. www.comp.lancs.ac. uk/computing/resources/ IanS/SE5 /syseng /sise/intro. ppt.

[22] Sommerville, I. (2001): Software Engineering, 6th ed. ISBN: 0-631- 21304-X, Addison-Wesley.

[23] Soriyan, H. (2004). A Conceptual Framework for Information Systems Development Methodologies For Education And Industrial Sector In Nigeria. PhD Thesis, Obafemi Awolowo University Ile-Ife.

[24] Soriyan, H. (2004). A Conceptual Framework for Information Systems Development Methodologies For Education And Industrial Sector In Nigeria. PhD Thesis, Obafemi Awolowo University Ile-Ife.

[25] Whitten, L., Bentley D., and Kevin, C. (2001), Systems Analysis and Design Methods. McGraw-Hill, Irwin.