A LITERATURE SURVEY OF COGNITIVE COMPLEXITY METRICS FOR STATECHART DIAGRAMS

Ann Wambui King'ori[1, 2], Geoffrey Muchiri Muketha[1] and Elyjoy Muthoni Micheni[3]

[1]School of Computing and Information Technology, Murang' a University of Technology, Kenya

[2]Department of Information Communication Technology, Nkabune Technical Training Institute, Kenya

[3]School of Business and Management Sciences, The Technical University of Kenya, Kenya

ABSTRACT

Statechart diagrams have inherent complexity which keeps increasing every time the diagrams are modified. This complexity poses problems in comprehending statechart diagrams. The study of cognitive complexity has over the years provided valuable information for the design of improved software systems. Researchers have proposed numerous metrics that have been used to measure and therefore control the complexity of software. However, there is inadequate literature related to cognitive complexity metrics that can apply to measure statechart diagrams. In this study, a literature survey of statechart diagrams is conducted to investigate if there are any gaps in the literature. Initially, a description of UML and statechart diagrams is presented, followed by the complexities associated with statechart diagrams and finally an analysis of existing cognitive complexity metrics and metrics related to statechart diagrams. Findings indicate that metrics that employ cognitive weights to measure statechart diagrams are lacking.

KEYWORDS

UML, Statechart diagrams, Software metrics, Cognitive complexity metrics, statechart complexity metrics

1. INTRODUCTION

Unified Modeling Language (UML) is a language that represents a software system visually [29]. The language is currently one of the most widely used for modeling of systems programs using object-oriented technology [6, 12, 17, 18]. UML diagrams can be divided into two categories, namely, structural UML diagrams that show how the system is structured, and behavioural UML diagrams that visualize the dynamic behavior of the system [7, 14, 25, 28].

Researchers have recognized the importance of UML models in today's software development. The benefits associated with modeling using UML language include reduction of the software development expenses, speeding up the process of the building system software, and development of good quality system programs [7].

A statechart is one of the pillar diagrams of UML used to model the dynamic nature of a system. A typical statechart diagram is composed of states, transitions, and events. These states are changed by events. Statechart diagrams have inherent complexity that keeps growing with age which may affect their cognitive effectiveness. Researchers agree that high cognitive complexity

DOI: 10.5121/ijsea.2019.10403

indicates poor design, which sometimes can be unmanageable thus leading to low- quality diagrams [2, 23]. Researchers have proposed numerous metrics that have been used to measure and therefore control the complexity of these diagrams. However, there is a lack of literature that documents the statecharts cognitive complexity.

This paper is, therefore, a literature survey of existing cognitive complexity metrics for UML diagrams and existing metrics for statechart diagrams. The survey is conducted to investigate if there are any gaps in the literature.

The rest of this paper is structured as follows: Section two presents a short overview of the basic concepts on statechart diagrams, section three presents the complexity of statechart diagrams, and sections four and five covers the existing cognitive complexity metrics for UML diagrams and existing metrics for statechart diagrams respectively. Section six presents the conclusions.

2. BASIC CONCEPTS OF STATECHART DIAGRAMS

Statechart diagrams capture the life cycles of objects, subsystems, and systems. The diagrams show states, transitions, events, and activities of a software system. They also describe the different states of an object and how internal and external events can change those states [13]. These events include received messages, the time elapsed, errors and conditions becoming true. A statechart diagram should be attached to all classes that have identifiable states and complex behavior. A statechart also shows different events that change the state and activities that take place in a particular state. Researchers have identified three key elements of statecharts diagrams namely; states, events, and transitions. Figure1 shows states, transitions, and actions performed when a card is swiped-in on an automated teller machine.

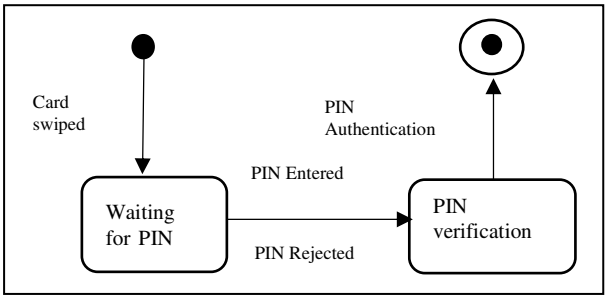


Figure 1. Statechart for an ATM

2.1. States

A state represents a discrete, continuous segment of time where the object's behavior will be stable. The object remains in a state until an event triggers a state change. In [26] a state is defined as a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for an event. The state of an object is dependent on how it interacts with other objects and the entry, do and, exit activities it is performing [26]. A state has five parts namely; state name, entry (action performed upon entry to a state), do activity, exit state (action performed on leaving a state) and deferrable triggers which are events that are postponed to be handled in another state as shown in Figure 2. A state is drawn with a round rectangle with one or more regions.

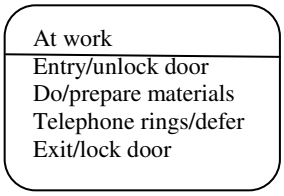


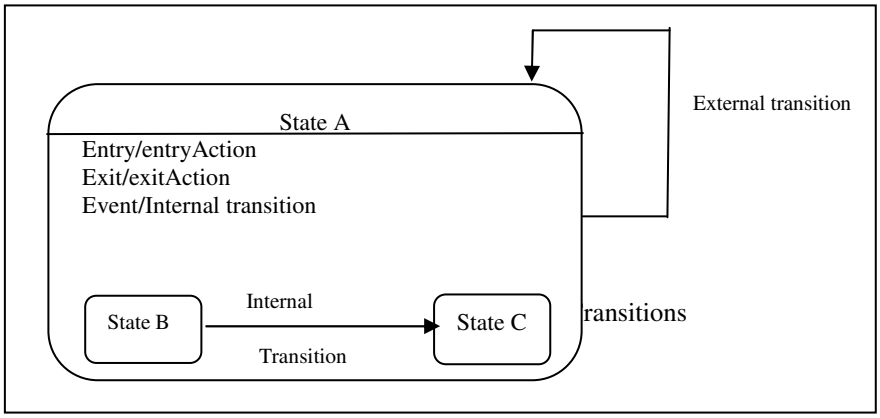
Figure 2. Parts of a State

2.2. Events

An event is a stimulus that triggers state changes. Events are representations of requests from other objects. In [13, 26] an event is defined as the specifications of noteworthy occurrence that has an allocation in time and space. Events trigger transitions in state machines. Events can be shown externally by transitions. Events are written simply as text strings. There are four types of event: 1) Call event – a call event is explained as a request for a specific operation to be invoked in an instance of the context class [13, 26]; 2) Signal event – a signal event represents the reception of a signal. A signal is modeled as a stereotyped class that holds the information to be communicated in its attributes; 3) Change event – a change event models an event that occurs when the guard condition becomes true as a result of a change in the value of one or more attributes; 4) Time event – a time event represents the elapsed time. The keyword when species a particular time at which the event is triggered; after keyword specifies a threshold time after which the event is triggered.

2.3. Transitions

A transition is a change from one state to another. A transition has five parts, namely, source state, event trigger, guard condition, action, and target state. Transitions in behavioral state machines have a simple syntax that may be used for external transitions or internal transitions as shown in Figure 3. A transition can either have zero or more events, zero or more guards and zero or more actions [26]. A transition is drawn with an arrow labeled with a guard or action to perform.



3. COMPLEXITY OF STATECHART DIAGRAMS

Complexity is the state of being difficult. Statechart diagrams have undesirable features that may affect their cognitive effectiveness. Cognitive effectiveness is the mental difficulties for

performing a task such as understanding code or a software system. UML understandability is influenced by the size and complexity of the diagrams [3, 5]. As the cognitive load of statechart diagrams increases, the task of the engineers in designing and validating the system becomes increasingly difficult. Statechart diagrams are associated with the following complexities: 1) Symbol Redundancy – symbol redundancy occurs when multiple graphical symbols are used to represent one semantic construct. Symbol redundancy in a modeling notation can confuse modelers as to which notation to use, leading to confusion concerning their referent semantic [1]; 2) Symbol Overload – symbol overload in UML means that one graphical symbol represents multiple semantic constructs. A construct overload occurs when one UML graphical construct represents two or more UML concepts. Symbol overload leads to confusion as to the precise meaning of a symbol leading to misunderstanding and misinterpretation on the actual meaning of the graphical symbols [1]; 3) Dual Coding – dual coding is the act of describing diagrams using text. The theory of dual coding states that using text to complement graphical symbols improves the cognitive effectiveness of a notation. Therefore, the text allows meaning to be created from the diagrams. However, the statechart diagrams notation does not utilize dual coding which affects the cognitive effectiveness of these diagrams [1]; 4) Inadequacy of UML Diagrams – UML diagrams are huge, complex and some UML notations are not differentiated enough [32]. Inconsistency and confusing diagrams in UML impact the effectiveness of compiling declarative knowledge into procedural knowledge of modeling an aspect of a system using UML [27]. Also, the number of concepts is not captured using UML notation which makes the learning of UML diagrams difficult [27]; 5) The ambiguity of UML Semantics – UML lacks formal semantics and some semantics are not precisely defined. Meanings are hidden under the diagrams which create ambiguities at the implementation level. The ambiguity of UML semantics is an inherent problem due to the unifying process which produces UML from many different object-oriented modeling techniques [27]. The ambiguity of UML semantics makes it hard for engineers to generate codes from models that make it difficult to learn and understand the diagrams; 6) A large number of UML Constructs – UML constructs serve as the building block of UML diagrams. UML diagram provides a large set of constructs and notations which are poorly defined. A large number of constructs confuse designers as to which notation to use during software development hence increasing the difficulty in learning UML diagrams [27].

4. EXISTING COGNITIVE COMPLEXITY METRICS FOR UML DIAGRAMS

Cognitive complexity measures take into account both the internal structures of software and input/output processes. Several researchers have attempted to study software metrics. The following section presents existing cognitive complexity metrics for UML diagrams.

(i) Cognitive Functional Size (CFS)

The Cognitive functional size (CFS) measure has been proposed for measuring the cognitive functional size (CFS) of software [30]. According to these authors, the cognitive complexity of software depends on three factors: internal architectural control-flows, input data, and output data. The CFS metric is defined as follows:

$$CFS= (N_i +N_o) *W_c$$

Where N_i is the number of inputs, N_o is the number of outputs, and W_c is the total cognitive weight of all control-flow blocks in the program. The W_c assigned to different control-flow structures is 1 for a sequence, 2 for a branch, 3 for a loop and 4 for a parallel structure.

CFS has been validated theoretically and with case studies [19]. It, however, excludes some essential details of cognitive complexity such as information that is contained in operators and

operands hence cannot be used to measure the cognitive complexity aspect of UML statechart diagrams. Also, it does not consider some unique features of the object-oriented paradigm such as inheritance [23].

(ii) Modified Cognitive Complexity Measure (MCCM)

The modified cognitive complexity measure (MCCM) was a modification of cognitive functional size (CFS) [20]. The author considered the number of operators and operands and the cognitive weights due to the basic control structures. The complexity due to operators and operands is defined:

$$S_{OO} = N_{i1} + N_{i2}$$

Where N_{i1} is the total number of occurrences of operators, N_{i2} is the total number of occurrences of Operands and S_{OO} is the total number of operators and operands.

The modified Cognitive Complexity Measure is computed by multiplying the S_{OO} and W_c. The formula is defined as follows:

$$MCCM = S_{OO} * W_c$$

MCCM is reasonable from the cognitive informatics works and has not been used for statechart diagrams measurement. The metric has been criticized for providing complexity values high in number [11, 23].

(iii) Cognitive Program Complexity Measure (CPCM)

The Cognitive Program Complexity Measure (CPCM) was proposed in line with the rules of cognitive informatics [21]. The metric depends on operands (total occurrences of inputs and outputs) and cognitive weights (weights of the basic control structures). CPCM is defined as:

$$CPCM = S_{io} + W_c.$$

Where S_{io} is the complexity due to the occurrences of input and output variables and W_c is the cognitive weight of all basic control structures. CPCM depends on operands only unlike MCCM which depends on both operators and operands.

CPCM has been criticized for being unclear and ambiguously interpreted [10, 23]. It has also been criticized for not including complexity due to operators is not included in its formulation [11, 24].

(iv) Cognitive Information Complexity Measure (CICM)

The proponents of Cognitive Information Complexity Measure (CICM) agree that there are two factors that lead to complexity weighted information count of software and basic control structures [15]. CICM is defined as:

$$CICM = WICS * W_c$$

Where WICS is the sum of the weighted information count of every line of code of a given software and W_c is the cognitive weight of the basic control structure.

Cognitive Information Complexity Measure remains in the software domain and has not been used for statechart diagrams measurement. The metric has been criticized for being difficult to compute especially where the software contains many lines of code [23].

(v) New Cognitive Complexity of Program (NCCoP)

The New Cognitive Complexity of Program (NCCoP) measure is based on data objects, internal behavior of the software, the operands, and the individual weight of BCSs of every line of code [10]. The measure is defined as:

$$NCCoP = \sum_{k=1}^{LOCs} \sum_{v=1}^{LOCs} Nv * Wc(k)$$

This means that NCCoP is a function of constant lines of codes, variables, and basic control structure weight.

The New Cognitive Complexity of Program has been criticized for being unclear and ambiguous in the process of counting the number of variables per line of code [10, 16].

(vi) Cognitive Complexity for Business Process Model (CCBP)

Cognitive complexity is a state of the mental burden that involves the ease or difficulty to perceive a given task [16]. The cognitive complexity for business process model (CCBP) extends the cognitive functional size measure (CFS) in the software engineering field to measure the cognitive complexity of business process models. The cognitive complexity for business process model (CCBP) metric is a function of the total input, output information flows and the total cognitive weight (Wc) of the structured activities in a process model [24]. These information flows are represented by input and output activities found within a BPEL process model. The input and output information represents the coupling-related information between the process model and its clients/partners while the weights are fixed values for the different control-flow structures. The weights represent the psychological effort needed by a designer to comprehend a control-flow block of the process model [24].

To calculate CCBP, a count of the input and output activities in the process model is obtained and then multiplied by the total cognitive weight (Wc) for all structured activities within the model. CCBP is defined as follows:

$$CCBP= (NOIA+NOOA)*Wc.$$

Where NOIA is the number of input activities, NOOA is the number of output activities and Wc is the total cognitive weight of all control-flow blocks within the model.

Cognitive Complexity for Business Process Model has been validated both theoretically and empirically. However, the metrics can only be applied to business processes hence cannot be used on the measurement of the cognitive complexity of statechart diagrams.

5. EXISTING STATECHART DIAGRAM COMPLEXITY METRICS

Several researchers have attempted to study the complexity of the statechart diagram. This section present metrics related to statechart diagrams.

(i) Size and Structural Complexity metrics for UML statechart diagrams

Genero et al. [8] proposed size and structural complexity metrics to measure the understandability of statechart diagrams. These include the number of entry actions (NEntryA), number of exit actions (NexitA) number of activities (NA), number of simple states (NSS), number of composite states (NCS), number of events (NE), number of guards (NG), and number of transitions (NT).
The NEntryA metric calculates the total number of entry actions in a state in the statechart diagram while the NexitA metric calculates the total number of exit actions performed each time a state is left. In addition, the NA metric counts the total number of activities in a statechart diagram, the NSS metric counts the total number of simple states in a statechart diagram, the NCS metric counts the total number of composite states in a statechart diagram, the NE metric counts the total number of events in a statechart diagram, the NG metric counts the total number of guard conditions in a statechart diagram while the NT metric counts the total number of transitions in a statechart diagram.

Genero et al. [8] also modified the McCabe's cyclomatic complexity metric by defining it as |NSS|-|NT|+2.

The metrics have been validated both theoretically and empirically. However, the metrics do not employ the cognitive weight aspect of software thus cannot be used to measure the cognitive complexity of statechart diagrams.

(ii) Cohesion and Coupling Metrics for Statechart Diagrams

Dalijeet & Lavleen [4] proposed a set of metrics for statechart diagrams. These metrics are used for analyzing the cohesion and coupling of statechart diagrams using slicing techniques.
To measure the cohesiveness of statechart diagrams, Dalijeet & Lavleen [4] proposed Average Cohesiveness of States (ACOS) metric. This is a cohesion metric for a state diagram which is computed by averaging cohesiveness values of all states. ACOS of a state diagram (SD) is defined as follows:

$$ACOS\ (SD) = (SsS\ COS(s) / |S|$$

ACOS of a state diagram is defined to be between 0 and 1. ACOS is 1 if the cohesion of each state in SD is 1. This means that all states are only one semantic. On the other hand, low ACOS suggests that each state has low cohesion.

To measure the coupling of statechart diagrams, Dalijeet & Lavleen [4], proposed the average number of similar states of states (ASSOS) metric. This is a coupling metric for a state diagram defined as the average number of similar states of each state in the state diagram. ASSOS of a state diagram (SD) can be formulated as follows:

$$ASSOS\ (SD) = (SsS\ |SS(s)|) / |S|$$

Where ASSOS (SD) can be more than or equal or 0.

The metrics are reasonable from the software perspective. However, the metrics do not employ the cognitive weight perspective thus cannot be used to measure the cognitive complexity of statechart diagrams.

6. CONCLUSIONS AND FUTURE WORK

Findings in this study indicate that statechart diagrams have undesirable features that may affect their cognitive effectiveness. The study also indicates that cognitive complexity metrics for UML

statechart diagrams are lacking. Each of the discussed existing cognitive complexity metrics has its limitations. The study shows that the proposed metrics related to statechart diagrams do not employ the cognitive weight aspect of software hence cannot be used to compute the cognitive complexity of UML statechart diagrams.

To address the lack of statechart complexity measures that employ the cognitive weight aspect, future studies should focus on defining new weight-based complexity metrics for statechart diagrams.

REFERENCES

[1] Anwer, S., & El-Attar, M. (2014). An evaluation of the statechart diagrams visual syntax. *In 2014 International Conference on Information Science and Applications (ICISA) (pp. 1-4).* IEEE.

[2] Briand, L. C., Bunse, C. & Daly, J. W. (2001). A controlled Experiment for Evaluating Quality Guidelines on Maintainability of Object Oriented Design. *IEEE Transactions on Software Eng*.2 (6): 513–530.

[3] Cruz-Lemus, J. A., Genero, M., Manso, M. E., Morasca, S., & Piattini, M. (2009). Assessing the understandability of UML statechart diagrams with composite states—A family of empirical studies. *Empirical Software Engineering*, *14*(6), 685-719.

[4] Daljeet, S., & Lavleen, K. (2012). Analyzing the Cohesion and Coupling of Statechart Diagrams using Program Slicing Techniques. *International Journal of Computer Science and Technology*,3, 69-72.

[5] Dori, D., Wengrowicz, N., & Dori, Y. J. (2014). A comparative study of languages for model-based systems-of-systems engineering (MBSSE). In *2014 World Automation Congress (WAC)* (pp. 790-796). IEEE.

[6] Fahad A. (2012). State Based Static and Dynamic Formal Analysis of UML State Diagrams. *Journal of Software Engineering and Applications*, 5, 483-491.

[7] Fitsilis, P., Gerogiannis, V. C., & Anthopoulos, L. (2013). Role of Unified Modelling Language in Software Development in Greece-results from an exploratory study. *IET Software*, 8(4), pp. 143-153.

[8] Genero, M., Miranda, D., & Piattini, M. (2003). Defining Metrics for UML Statechart Diagrams in a Methodological way. *In International Conference on Conceptual Modelling*, (pp.118-128).

[9] IEEE Standard 1061 (1992). Standard for a Software Quality Metrics Methodology. *Institute of Electrical and Electronics Engineers*. New York.

[10] Jakhar, A.K & Rajnish, K. (2014). A new cognitive approach to measure the complexity of software's. *International Journal of Software Engineering & its Applications*, vol. 8, no. 7, pp. 185-198.

[11] Jakhar, A. K., & Rajnish, K. (2015). Measurement of complexity and comprehension of a program through a cognitive approach. *International Journal of Engineering-Transactions B: Applications*, *28*(11), 1579-1588.

[12] Jamal, M., & Zafar, N. A. (2016). Formalizing structural semantics of UML 2.5 activity diagram in Z Notation. In *2016 International Conference on Open Source Systems & Technologies (ICOSST)* (pp. 66-71). IEEE.

[13] Jama, O.M., (2009). *A Case Study on Evaluating UML Modelling in Software Testing (Master's thesis, University of OSLO).*

[14] Kumar, A., & Khalsa, S. K. (2012). Determine cohesion and coupling for class diagram through slicing techniques. *IJACE*, *4*(1), 19-24.

[15] Kushwaha, D. S. & Misra, A. K., (2006). Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties. *ACM SIGSOFT Software Engineer Notes*, 31(1), 1–6.

[16] Maheswaran, K., & Aloysius, A. (2017). An Analysis of Object Oriented Complexity Metrics. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *2*, 768-775.

[17] Maylawati, D. S., Darmalaksana, W., & Ramdhani, M. A. (2018). Systematic design of expert system using unified modelling language. In *IOP Conference Series: Materials Science and Engineering* (Vol. 288, No. 1, p. 012047).

[18] Miles, R. & Hamilton, K. (2006). *Learning UML 2.0*. "O'Reilly Media, Inc.".

[19] Misra, S. (2004). Evaluating cognitive complexity measure with Weyuker's properties. *3rd IEEE International Conference on Cognitive Informatics (ICCI'04)*: 103-108.

[20] Misra, S. (2006). Modified cognitive complexity measure.*LNCS* 4263: 1050-1059.

[21] Misra, S. (2007a). Cognitive program complexity measure. *6th IEEE International Conference on Cognitive Informatics*: 120-125.

[22] Misra, S. (2011). Cognitive complexity measures: An analysis. *Modern Software Engineering Concepts and Practices: Advanced Approaches,* (pp. 263-279), IGI Global.

[23] Misra, S., Adewumi, A., Fernandez-Sanz, L., & Damasevicius, R. (2018). A Suite of Object Oriented Cognitive Complexity Metrics. *IEEE Access*, 6, 8782–8796.

[24] Muketha, G.M. (2011). *Size and complexity metrics as indicators of maintainability of business process execution language process models* (Doctoral dissertation, Universiti Putra Malaysia).

[25] OMER, O. S. D., & Sahraoui, A. E. (2017). From Requirements Engineering to UML using Natural Language Processing–Survey Study.

[26] Rambaugh, J., Jacobsen, I. & Booch, G. (2005). The Unified Modelling Language Reference Manual, second Edition. Pearson Higher Education.

[27] Siau, K., & Loo, P.P. (2006). Identifying difficulties in learning UML. *Information Systems Management*, 23(3), 43-51.

[28] Sikka, P., & Kaur, K. (2016). Mingling of Program Slicing to Designing Phase. *Indian Journal of Science and Technology*, *9*(44).

[29] UML, O. (2012a). Information technology-Object Management Group Unified Modelling Language (OMG UML), Infrastructure.

[30] Wang, Y. & Shao J., (2003). A new Measure of Software Complexity Based on Cognitive Weights. *Journal of Electrical and Computer Engineering*, 28(2), 69-74.

[31] Wang, Y. (2004). On the Cognitive Informatics Foundations of Software Engineering. *In Proceedings of the Third IEEE International Conference on Cognitive Informatics*, pp. 22-31.

[32] Zafar, N. A. (2013). Model analysis of equivalence classes in UML events relations. *Journal of Software Engineering and Applications*, *6*(12), 653.

AUTHORS

Ann Wambui King'ori is an ICT Assistant Lecturer at the Department of Information Communication Technology at Nkabune Technical Training Institute, Kenya. She earned her Bachelor of Technology Education (Computer Studies) from the University of Eldoret, Kenya in 2014. She is currently pursuing her MSc. in Information Technology at Murang'a University of Technology, Kenya. Her research interests include software metrics, software quality, and business intelligence.

Geoffrey Muchiri Muketha is an Associate Professor and Dean of the School of Computing and Information Technology, Murang' a University of Technology, Kenya. He received his BSc. in Information Science from Moi University in 1995, his MSc. in Computer Science from Periyar University, India in 2004, and his Ph.D. in Software Engineering from Universiti Putra Malaysia in 2011. He has wide experience in teaching and supervision of postgraduate students. His research interests include software and business process metrics, software quality, verification and validation, empirical methods in software engineering, and component-based software engineering. He is a member of the International Association of Engineers (IAENG).

Elyjoy Muthoni Micheni is a Senior Lecturer in Information Systems in the Department of Management Science and Technology at The Technical University of Kenya. She holds a Ph.D. (Information Technology) from Masinde Muliro University of Science and Technology, Master of Science (Computer Based Information Systems) from Sunderland University, (UK); Bachelor of Education from Kenyatta University; Post Graduate Diploma in Project Management from Kenya Institute of Management. She has taught Management Information System courses for many years at the University level. She has presented papers in scientific conferences and has many publications in refereed journals. She has also co-authored a book for Middle-level colleges entitled: "Computerized Document Processing". Her career objective is to tap computer-based knowledge as a tool to advance business activities, promote research in ICT and enhance quality service.