# DEVELOPMENT OF COMMUNICATING STREAM X-MACHINE TOOL FOR MODELING AND GENERATING TEST CASES FOR AUTOMATED TELLER MACHINE

Bashir Adewale Sanusi[1], Emmanuel Ogunshile[1], Mehmet Aydin[1], Stephen Olatunde Olabiyisi[2] and Mayowa Oyedepo Oyediran[3]

[1]Department of Computer Science and Creative Technologies, University of the West of England, Bristol, United Kingdom
[2]Department of Computer Science, Ladoke Akintola University of Technology, Ogbomoso, Nigeria
[3]Department of Computer Sciences, Ajayi Crowther University, Oyo, Nigeria

***ABSTRACT***

*The improvement of this paper takes advantage of the existing formal method called Stream X-Machine by optimizing the theory and applying it to practice in a large-scale system. This optimized formal approach called Communicating Stream X-Machine (CSXM) applied in software testing based on its formal specifications to a distributed system as it points out its advantages and limits of the use of the existing formal methods to this level. However, despite the tremendous works that has been done in the software testing research area, the origin of bugs or defects in a software is still cost and takes more time to detect. Therefore, this paper has proven that the current state of art challenge is due to that lack of a formal specification of what exactly a software system is supposed to do. In this paper, CSXM principles was used for the development of Automated Teller Machine (ATM) given formal specification which outputs conforms with the implementation. Moreso, the computational strength of Remote Method Invocation (RMI) network interface in Java programming was used to provide communication between the stand-alone systems i.e., the client (ATM) and server (Bank) in the context of this paper. The results of this paper have been proven and helps software developers and researchers takes early action on bugs or defects discovered by software testing.*

***KEYWORDS***

*Formal Method, Software Testing; Stream X-machine; Communicating Stream X-Machine; Software Testing; Distributed System; Formal Specification; Defects; Automated Teller Machine; Remote Method Invocation; Java Programming Language.*

## 1. INTRODUCTION

In computational term, testing attempts to achieve correctness by detecting all the faults that are present in an implementation, for the errors to be removed. Nevertheless, software defect is referred to as a flaw, fault or failure in a computer system or program which gives an unexpected or incorrect result [1]. It gives either an incorrect, or unexpected result, and behaves in unintended ways. The unexpected result is identified during software testing and marked as a defect [1]. Stream X-Machine (SXM) testing methodology is a complete functional testing approach to software and hardware testing that exploits the scalability of the SXM model of computation [2]. SXM testing method provides repeatable and strong guarantees of functional correctness, up to a specification [3]. To model systems composed of communicating agents and introduction of stand-alone SXM models, communication is suitable by exchanging messages between components processing functions [4]. In addition, the design of a Communicating X-machine system (CXM) is referred to as the graph whose nodes are the components and edges are simply the communication channels among all. Sequel to the issue of integrating a society of X-machine into a communicating system for the goal of building large-scale software systems which fulfil their requirements, several classes of CXM models have been presented [5]. However, CXM is a formal model which ease a regimented development of large-scale systems. Nevertheless, the optimization of the existing formal method in use called SXM has leads to theory of the Communicating Stream X-Machine (CSXM) which has been proven and best suited the specification of a distributed system. With the ever-expanding areas of applications today including embedded and real-time systems, safety-critical systems, service-oriented architectures and so on, it is often easy to lose sight of the essential similarities that exist among all of these systems using formal methods. But the critical issue is the relationship between the proposed solution and the understanding of the problem's originator as to whether the proposed solution does, in fact provide the desired answer. Therefore, in this paper the strength of the CSXM was used to specify a distributed system called Automated Teller Machine (ATM) which was correctly implemented and satisfies the challenge of software testing.

## 2. Related Work

[6], transformed X-machine specification written in X-Machine Language (XML) and which is automatically converted into an executable java code. This research was unable to write test cases, and it is limited to the theory of standalone X-machine components. [7], reports on the development and formal verification of CompCert, i.e., a compiler from Clight which is referred to as the subset of the C programming language to PowerPC assembly code. This research used the Coq proof assistant both for programming the compiler and for proving its correctness. Also, it stated that the verified compiler is useful in the context of the critical software and its formal verification.

[8], center the research on providing tool support for business-level, example-based specifications which are mapped to the browser level for automatic verification. This research allows refactoring support for the evolution of existing browser-level tests into business-level specifications, and it provides feedback on coverage as the resulting business rule tables may be incomplete, contradictory, or redundant. [9], stated formal verification are used for exhaustive investigation of the system space which ensures that undetected failures in the behavior are excluded. The research constructs the system incrementally from subcomponents, based on the software architecture. According to this study, developing a safe multi-agent robotic system to ensure the correctness properties of safety and liveness was a challenge. In conclusion, the development approach allows for formal verification during the specification definition.

According to [10], the software testing may consume 35% to 40% of a software development budget. However, the manual and automated testing methods seems complementary to each other. In this study, it is stated that the purpose of testing is to find out defects or bugs, the causes, and approaches in which this fault can be fixed as early as possible. It is therefore stated that the testing requires more project effort and time than any other software development activity which needs a suitable strategy to make the testing successful.

[11], analyzes three states of the art formally verified implementations of distributed systems which includes Iron-Fleet, Verdi, and Chapar. This research sees through the code review, testing and found a total of sixteen bugs in which some produces serious consequences, including

crashing servers, returning incorrect results to clients, and invalidating verification guarantees. Therefore, this research develops a testing toolkit called PK, which focuses on testing these parts and is able to automate the detection of thirteen out of sixteen bugs.

[12], share their experience and discuss the open problems or challenges that the software testing, verification, and compiler development presents. Although, the aim was to deliberate on new ideas on how to approach these problems. It is concluded that there is no universal appropriate name for this field. However, the compiler explains most of the work in this field and is not difficult to explain to people but has the barrier of excluding some subjects such as debuggers. Due to this reason, the research decided to keep the name compiler testing and verification for future research works.

[13], developed an automatic Java X-Machine testing tool for software development. This research focuses on addressing the software complexity and changing the software developers' expectations with the motive of reducing the amount or cost in detecting defects in software systems. It was concluded that this research could not generate test cases automatically.

[14], stated that for a compiler correctness theorem to assure complete trust, then such theorem must reflect the reality of how the compiler will be used. Although, the variation of theorems, stated in remarkably different ways, develops questions about what researchers meant by a compiler is correct. Therefore, this study developed a framework with the idea to understand compiler correctness theorems in the presence of linking and applying it to understanding and comparing this diversity of results. Hence, this research did not only focus on the strengths and weaknesses but also gain insight into what should be expected from compiler correctness theorems of the future research.

[1], conducted experiments on publicly available bug prediction dataset that is a repository for most open-source software. This research used Genetic algorithm to extract relevant data features from the acquired dataset and machine learning algorithms was used to predict defect in software system. [15], proposed an enhanced fault-detection W method for increasing software reliability in safety-critical embedded systems. The research stated the testing time of the proposed method takes much time than the W method during the software testing.

[16], stated software defect prediction is one of the most encouraging exercises of the testing phase of the software development life cycle. In this case this research created a framework to anticipate the modules that deformity inclined the software quality. Nonetheless, GA was used to extract the relevant features from publicly available data sets to eliminate the possibility of overfitting and the relevant features were classified into defective or non-defective. In conclusion, the outcome indicated that ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK and LUCENE has the accuracy, precision, recall and the f-score of 86.93, 53.49, 79.31 and 63.89% respectively, 83.28, 31.91, 45.45 and 37.50% respectively, 83.43, 57.69, 45.45 and 50.84% respectively and 91.30, 33.33, 50.00 and 40.00% respectively.

[17], study and evaluate a narrative verification approach based on Bounded Model Checking (BMC) and Satisfiability Modulo Theories (SMT) to verify C++ source codes. This research verification approach analyses bounded C++ source codes by encoding into SMT various sophisticated features that the C++ programming language offers which includes inheritance, templates, polymorphism, exception handling, and the standard template libraries. Nevertheless, the research compares Efficient SMT BMC (ESBMC) to The Low-Level BMC (LLBMC) and DIVINE, which are the state-of-the-art verifiers to check C++ source code directly from the LLVM bitcode. It is concluded in this research that ESBMC can handle a wide range of C++ source codes, presenting a higher number of correct verification results.

Therefore, in this paper, having reviewed some existing papers on the previous methods that researchers have applied in the software testing stage of the software development lifecycle, but this area of research remains researchable and continuous research area. Formal methods as being proven lately by researchers in specifying how a system should work. Hence, the SXM was improved on by using the Communicating Stream X-Machine (CSXM) theory to prove that the specification conforms with the implementation of the said case study and will be used in future research to assure the correctness, testing, and verification of a compiler design by integrating the strength and computational use of machine learning algorithms so as to construct test programs which will be used to determine whether a compiler behaves correctly.

## 3. METHODOLOGY

As the Software Development Life Cycle (SDLC) is said to be the most used and oldest formalized framework for constructing distributed systems [18]. The strength of this methodology framework was used to build the large-scale distributed system (ATM) in this paper. There are different models in SDLC but the choice of waterfall model in this paper was its strength of step-by-step flow in respect to its importance or an organized process that makes sure every stage is followed carefully before moving on to the next steps. As a result of this, the waterfall model was adopted in this paper to divide the software development work into sequential steps and smaller process to optimize the design and specification of the SXM. The strength of this methodology has helped this paper to build and deliver a correct and stable system. It also creates a clear understanding of the task ahead as stated in the last paragraph of section 2 of this paper which enables a better estimate and identify errors earlier i.e., the specifications conforming with the implementation. Nevertheless, advancement in computing capabilities puts higher demand on the software system and the developers. This raises the issues of cost, delivery of faster software, and conforming to the needs of the end users. Therefore, SDLC strength was used in this paper to measure the correctness, testing which identify inefficiencies and verifying them in the development process which helps in fixing the errors and run smoothly. Therefore, the case study ATM was designed and implemented using the CSXM formal theory written in Java programming language to test and proof the correctness of the developed system.

### 3.1. Steps followed in this Paper

The following are the stages that was adopted in this case study. Figure 1 represents the diagrammatic design of the developed system that was used in this paper.

- Requirement Gathering and Analysis: this phase is the definition of the requirements considered in planning which established what the software system is supposed to do as specified and its requirements. In this paper, ATM was the software constructed which required the ability to make transactions such as deposit, withdrawal e.t.c. However, this requirement includes the definition of the resources used to build this software which is use of CSXM specification to develop the software system. Hence, CSXM theory is the requirement used in this process.
- System Design: This phase models the functioning of the software system i.e., how the system is working. Also, the CSXM specification was written in Java Programming language which defined the way end users interacted with the software system and how the application responded to the end users' input. In this paper, the system was developed to run on MS Windows platform due to the methods used in solving the problems in the case study specified. With respect to the strength of Java programming language, the

Remote Method Invocation (RMI) was the method defined in the software system which enables the communication between the SXMs.

- Software Development: This phase is the system coding which is used as the access control in the software system. In this paper, this stage was used to track changes to the code, helps to verify the system by ensuring the specification conforms with this implementation. Nevertheless, finding, and fixing errors is an area generating interest in the field of software engineering which cannot be left out in the development process such as generating test cases or compilation of the code to ensure the software system runs as specified. This paper utilizes the computational power of the formal method (CSXM) which leads to the next stage of this life cycle.
- Software Testing: In SDLC, testing stage is a critical phase of the development process of a software system. Nonetheless, the choice of the formal method (CSXM) used in this paper was to explore one of the strengths which involves testing the correctness of the software system. In this paper, as the entire system was tested for any defects or bugs and failures so also verified by ensuring the specification conforms with the implementation.
- Maintenance: Over the decades, the cost and time of software testing continues to be a researchable area among researchers and everyday concern among software developers. At this phase, the software system is completed and being used by the end users. However, the end users may or may not discover defects that was not found during the testing stage which makes this stage an important phase in the SDLC. Therefore, the bugs found during this process are resolved and generate the new development cycles.

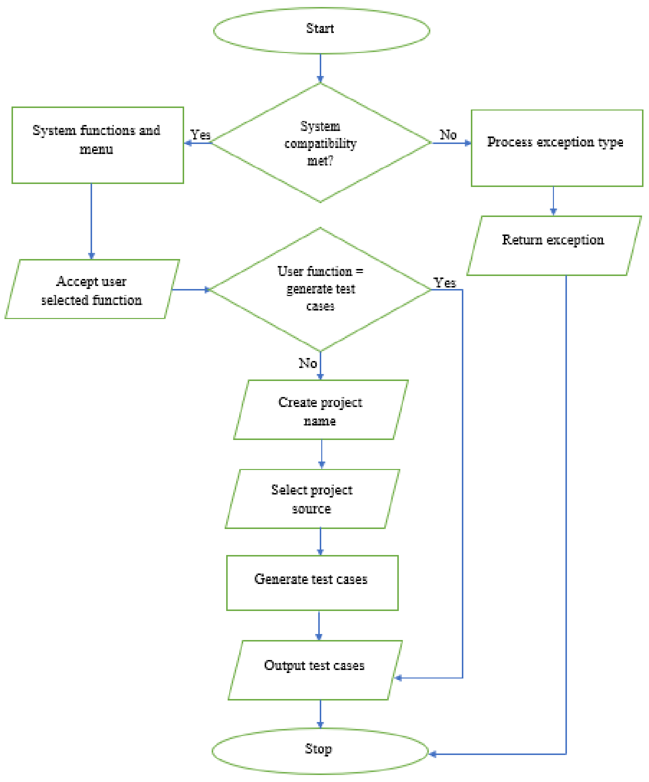


Figure 1. The Flowchart of the Developed System

## 3.2. Stream X-machine

In particular, the benefit of this model is that it permits a system to be driven, with extra care, through its states and transitions while noticing the results at each progression. These are observer esteems that ensure that specific capacities were executed on each progression [4]. Because of this explanation, complex software systems might be deteriorated into a pecking order of SXM, planned in a hierarchical way and tried in a base up manner. However, SXM is an eight tuple which are as follows;

$$Machine = (Q, \sum, \Gamma, M, \Phi, q_o, F, m_o) \quad (1)$$

Where;
$Q$: is the finite set of states
$\sum$: is the finite set of input symbols
$\Gamma$: is the finite set of output symbols
$M$: is a (possibly infinite) set called memory
$\Phi$: is the finite set of distinct processing functions; a processing function is a non-empty (partial) function of type $\mathrm{M} \times \sum \rightarrow \Gamma \times \mathrm{M}$
$q_o \in Q$: is the Initial state
$F$: is the (partial) next-state function, $\mathrm{F}: \mathrm{Q} \times \Phi \rightarrow Q$
$m_o \in M$: is the initial memory

Beginning from the initial state $q_o$ alongside the initial memory $m_o$, an input symbol $\sigma \in \sum$ activate a function $\varphi \in \Phi$ which in turn generate a transition to a new state $q \in Q$ and a new memory state $m \in M$. However, sequence of transitions generated by the stream of input symbols is called the computation. Therefore, the output of a computation is the sequence of results generated by the sequence of transitions.

## 3.3. Communicating Stream X-machine

In this paper, the CSXM systems model was adopted, and it is further reviewed below. Therefore, CSXM with $n$ x-machine components are a triplet tuple [5];

$$Machine_n = (R, MAT, C^0) \quad (2)$$

Where;

i. $R$ is the set of $n = |R|$ x-machine components of the system of the form $V_i = (\Lambda_i, IN_i, OUT_i, in_i^0, out_i^0) \forall\, 1 \leq i \leq n$. Such x-machine components of the system are called the Communicating X-Machine (CXM). $\Lambda_i$ in the definition stated above refers to a SXM with memory $M_i$. The $IN_i$ and $OUT_i$ directly correspond to the values that can be communicated by input and output ports of the $i$th CSXM such that $IN_i, OUT_i \subseteq M_i \cup \{\lambda\}$ and $\lambda \notin M_i$. The symbol $\lambda$ is simply used to indicate that a port is empty and the initial values of the x-machine ports are set to $in_i^0$ and $out_i^0$.

ii. $MAT$ is simply referred to as the set of matrices of order $n \times n$ to form the values of the matrix variable which supposed to be used for creating communication amongst the x-machine components. Therefore, for any $C \in MAT$ and any pair of x-machine such that $i, j$ are the data value stored in $C|i, j|$ shows at most one message that is being processed from the memory $M_i$ of x-machine $V_i \in R$ to the memory $M_j$ of x-machine $V_j \in R$. Thus, each element of the matrix $C|i, j|$ can be considered as a temporary buffer variable where the property $IN_i \subseteq M_i \subseteq OUT_j$ holds.

iii. Basically, all messages that are sent from the CXM $V_i$ that is x-machine $\wedge_i$ and $V_j$ which is x-machine $\wedge_j$ are data values from the memory $M_i$ and $M_j$ respectively. The $\lambda$ symbol in the matrices is used to specify that there is no message, where the @ symbol is simply used for specifying a channel that is not going to be used (an x-machine communicating with itself disallowed). The individual elements of the matrices are derived from the machine memory $M \cup \{\lambda, @\}$, where;

$$M = \bigcup_{i=1}^{machine} M_i \ and \ \lambda, @ \notin M \qquad (3)$$

iv. $C^0$ simply defines the initial communication matrix as $C^0|i,j| = \lambda$ assuming a valid communication between the x-machine $V_i$ that is x-machine $\wedge_i$ and $V_j$ which is x-machine $\wedge_j$ is allowed; otherwise, the initial matrix is defined as $C^0|i,j| = @$ so as to indicate that the communication between the two x-machines $i \ and \ j$ are disallowed. Nevertheless, the matrix $C^0|i,j| = @$ shows that an x-machine communicating with itself is definitely not allowed.

v. Generally, the $i$th CXM component can only read from the $i$th column and then write to the $i$th row of the communication matrix.

Also, for any $C \in MAT$, any value $x \in M$ and any pair of indices $1 \leq i,j \leq n$, with $i \neq j$.

i. If $C|i,j| = \lambda$ an output variant of $C$, denoted by $C_{ij} \Leftarrow x$ is defined as;

$$(C_{ij} \Leftarrow x)[i,j] = x \qquad (4)$$
$$(C_{ij} \Leftarrow x)[k,m] = C[k,m] \ \forall \ (k,m) \neq (i,j) \qquad (5)$$

ii. If $C|i,j| = x$ an input variant of $C$, denoted by $\Leftarrow C_{ij}$ is defined as;

$$(\Leftarrow C_{ij})[i,j] = \lambda \qquad (6)$$
$$(\Leftarrow C_{ij})[k,m] = C[k,m] \ \forall \ (k,m) \neq (i,j) \qquad (7)$$

As stated in equation (4), (5), (6), and (7) above, simply denotes several acceptable transitions from one matrix to another. Generally, CXM is a five tuple;

$$V = (\Lambda, IN, OUT, in^0, out^0) \qquad (8)$$

Recall, SXM in equation (1) $Machine = (Q, \sum, \Gamma, M, \Phi, q_o, F, m_o)$ which is equivalent to $\Lambda$ and stated below;

$$\Lambda = (Q, \sum, \Gamma, M, \Phi, q_o, F, m_o) \qquad (9)$$

Where;

i. $IN$ and $OUT$ has been defined within equation (2) above
ii. $\sum$ and $\Gamma$ are the finite set of input and output symbols respectively
iii. $Q$ is the finite set of states of each x-machine component gathered into a communicating system must be partitioned as $Q = Q^{'} \cup Q^{''}$ where $Q^{'}$ corresponds to the processing states in each x-machine component in the communicating system and $Q^{''}$ is the set of

communicating states corresponding to the central medium where all the $n$ x-machine components have been integrated and where $Q^{I} \cap Q^{II} = \emptyset$ holds. Therefore, this implies that for each $q^{I} \in Q^{I}$ in each x-machine component, the functions emerging from $q^{I}$ are processing functions. For instance, in the state $q^{I}$ different functions can be triggered, in this scenario one of them is arbitrarily chosen or if no function can be applied the entire CXM system halts. However, if the machine is in state $q^{II} \in Q^{II}$ then all the functions emerging from state $q^{II}$ are communicating functions. Otherwise, while the machine is in state $q^{II}$, if different functions can be applied then one of them is arbitrarily chosen, else if this is not the case then the machine simply does not change it current state and would have to wait until one of such functions can be applied.

iv. $M$ a set or possibly infinite set called the memory

v. The type of the machine is defined as a set $\Phi = \Phi^{I} \cup \Phi^{II}$ where $\Phi^{I}$ is called the set of processing function and $\Phi^{II}$ is the set of communicating functions and $\Phi^{I} \cap \Phi^{II} = \emptyset$. Notably, each element $\phi^{I} \in \Phi^{I}$ is a relation (partial function) of the type below;

$$\phi^{I}: IN \times M \times OUT \times \Sigma^* \rightarrow \Gamma^* \times IN \times M \times OUT \qquad (10)$$

## 4. Results and Discussion

In this paper, to assure the correctness, testing, and verification of the formal method used called the CSXM, a software system was developed called Automated Teller Machine (ATM) as it is one of the good and reliable examples of a distributed system. The ATM called client in this in this paper is the combination of computer terminal which communicates with one another to a bank's central computers also referred to as server in the context of this paper. The software system specifications in this paper enables end users to perform transactions such as login (security check), deposit, withdraw, inquiry, statement, and account details.

### 4.1. Developed System

The understanding of SXM and CSXM from theory to practice was used in a real-world scenario to proof the correctness and to test whether the ATM specification conforms with the developed implementation as constructed in this paper. However, as stated in equation (8) which includes the SXM, input, and output port. The SXM specified in this paper represents the overall specification of the said case study (ATM) as shown in figure 2 below. The figure 2 represents the state diagram of the overall case study (ATM). In formal methods, the said theory can be represented either by mathematical notation as stated in section 3.1 above or by state diagram which is the diagrammatical representation of the specified mathematical notations. From figure 2, the states include start, ATM_OutofSource, Insert_Card, and Continue_Transaction, where start is the initial state. Also, the transitions are ATM_InUse, Card_Insert, ATM_Not_Available, and Transaction_Aborted. The transition functions enable the states to move from one to another. Nevertheless, the strength of memory included in this model enables each customer stored information to be assessed when their information is being verified from the server. In this paper, the distributed banking system consist of some ATM referred to as client and a server called the bank which communicates via Java Remote Method Invocation (RMI). The server controls all users account information where an end user can use the following operations at an ATM.

i. void deposit (int acnt, int amt): this simply referred to when the operation increases the balance of end user account acnt by amt and returns nothing.

ii. withdraw (int acnt, int amt): this simply referred to when the operation decreases the balance of end user account acnt by amt and returns nothing.
iii. Float inquiry (int acnt): this referred to when the operation returns the balance of the end user account acnt.
iv. GetStatement (Date from, Date to): this also referred to when the operation returns a statement object of the transaction history of the said account acnt.

Nevertheless, the client (ATM) was used to initiate an end user operation by calling a remote method on the bank server to execute the specified procedure such as withdraw with its parameters as presented in Fig. 3 and Fig. 4 represents server (bank) authentication by validating the end users' card and pin before a transaction can be performed. Figures 3 and 4 are specified as two different SXM which are communicated with the RMI and give the representation of the communicating stream x machine (CSXM) as shown in Figure 5 below. In these figures the states are represented in oval shapes while the transitions are represented in arrows.

Furthermore, the concept of SXM and CSXM are discussed in section 3 of this paper. However, this paper is incomplete without giving details about the RMI. Java programming language has the strength which enables software developers write large scale object-oriented system of this manner where objects on different computers i.e., SXMs can interact in a distributed network. Regardless, RMI was used in this paper because it has the ability to pass one or more objects along with the request which suites the aim of this paper. This object includes information that change the service that is performed in the remote computer.

Moreso, in this paper when a user at the client (ATM) fills out an expense account, the SXM specification interacting with the user was communicated using RMI with a SXM specification in the server (Bank) that had the latest policy about the expense reporting. As a result of this, the program send back an object and associated method information that enables the client (ATM) program to screen the end users expense account data in a way that was consistent with the bank policy.

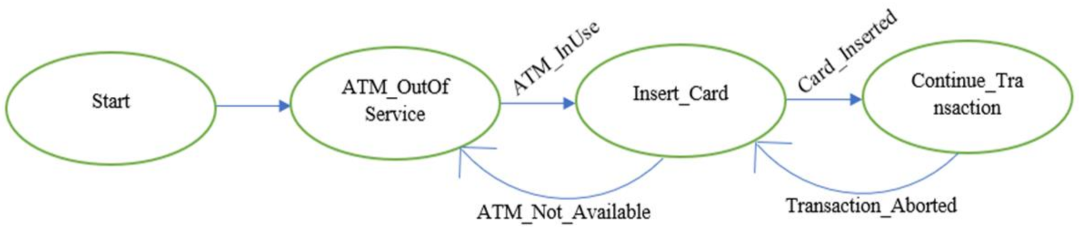


Figure 2. State Diagram of the Overall Case Study (ATM)

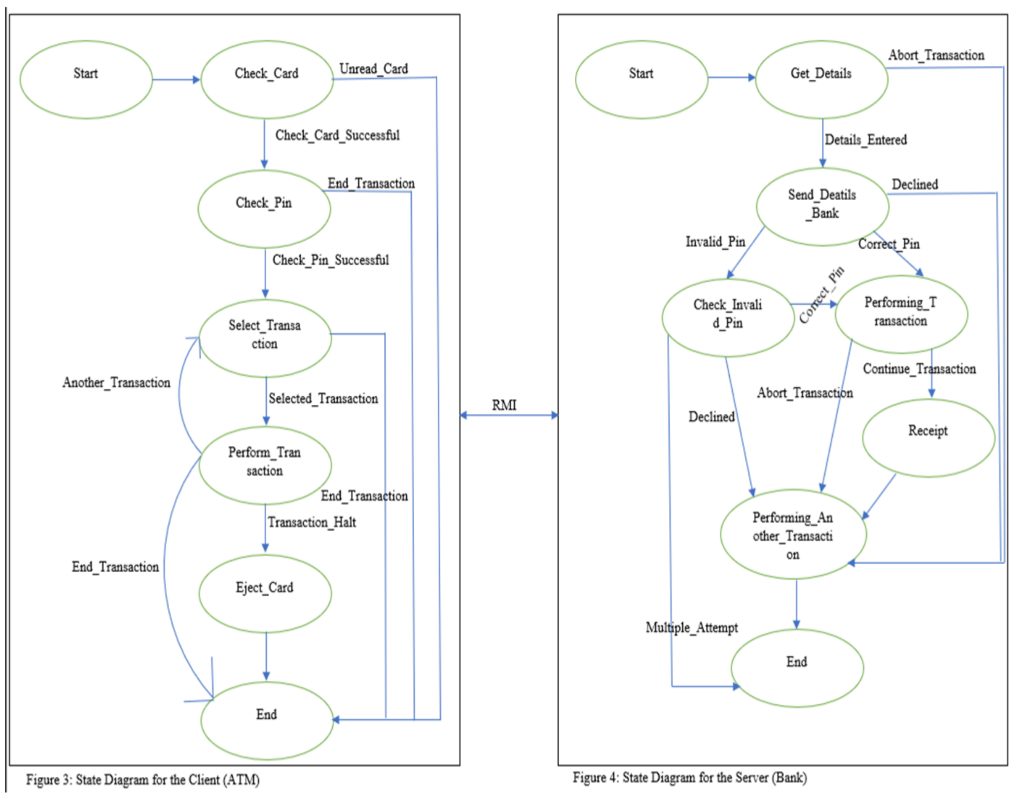


Figure 5. State Diagram of the CSXM.

The formal method (CSXM) used in this paper enables the end user and the bank save time by locating mistakes early which gives the advantage of correct system i.e., the specification meeting the implementations. However, whenever the bank policy changed, it would require a change to the SXM specification in only one computer. The RMI used in this paper was implemented as three layers when communicating with the SXMs i.e., client (ATM) and the server (Bank) as represented in Figure 6. These layers are as follows;

i. Stud: the stud used here is called the proxy that appears to the calling program which is then the program that is later called for the service.
ii. Remote Reference Later (RRL): the RRL was used to determine whether the request is to call a single remote service or more when considering a multicast.
iii. Transport Later: the transport connection layer was used to set up and manage the end user's request.

Hence, a single request was transmitted down through the layers on one machine and up through the layers at the other end. Also, the output of the generated test cases is presented in Fig. 7 below.

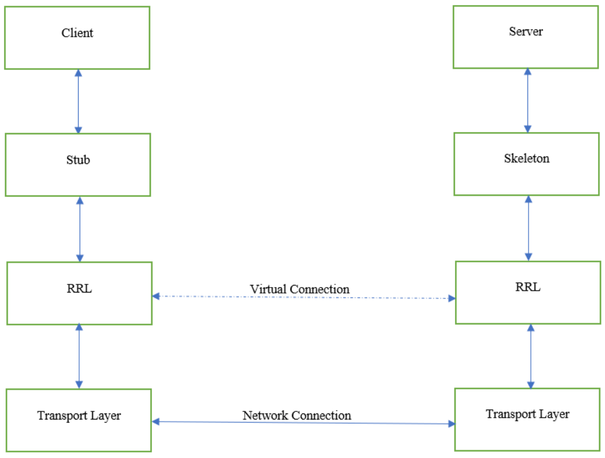
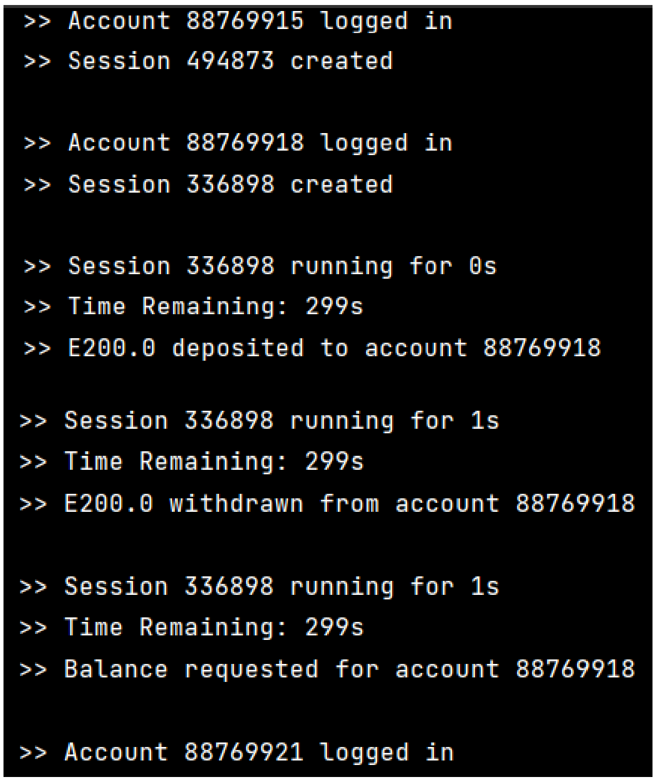


Figure 6. Architecture of the RMI.



```
>> Account 88769915 logged in
>> Session 494873 created

>> Account 88769918 logged in
>> Session 336898 created

>> Session 336898 running for 0s
>> Time Remaining: 299s
>> E200.0 deposited to account 88769918

>> Session 336898 running for 1s
>> Time Remaining: 299s
>> E200.0 withdrawn from account 88769918

>> Session 336898 running for 1s
>> Time Remaining: 299s
>> Balance requested for account 88769918

>> Account 88769921 logged in
```

Figure 7. Output of the Generated Test Cases.

### 4.2. Evaluation of the Developed Tool

In this paper, the developed tool was evaluated check for the correctness of the system by checking the specification if it conforms with the implementation. Furthermore, during this process some parts of the code are indicated which needs refactoring and the generated report are acted upon to assure the correctness of the software system. Moreso, the source code changes are monitored, and the functionality of this tool checks up to 500,000 lines of code. Therefore, the choice of the black box testing method was selected simply because it evaluates the functionality of an application or tool without having investigated its internal structure. Software testers and programmers were considered in the testing phase without disclosing the structure and design of the source code.

## 5. CONCLUSION

In conclusion, software testing in SDLC still continues to be a great research interest among researchers and software developers. However, many works have been done but still removing defects or bugs during and after development process is crucial and cannot be left out. First version of this paper was also based on using the formal method (SXM) strengths to ensure all specified in the software was implemented correctly. Optimization and improvement in the SXM extend the interest of using formal methods specification for future software testing. CSXM theory was used and practiced in this paper because of its advantage over the previous techniques of solving large-scale system problems. Nevertheless, this is not the end of this great research area as more work are proposed for future problems. Notably, the trending research as established the area compiler design where bugs not only cause unintended behavior with possibly severe consequence but also make software debugging more difficult because it is not easy to detect whether the bugs actually come from the compiler used to compile the source code or the source code itself. This as also been observed in SDLC process to avoid spending more time in checking for errors in the source code. One of the challenges is the lack of a formal specification of what exactly a compiler is supposed to do. Therefore, the future research will focus on reviewing the computational power of machine learning algorithms and integrating it with the strengths of CSXM to assure the correctness, testing, and verification of a compiler design.

## ACKNOWLEDGMENT

First, we would like to give our sincere gratitude to Almighty God for His supports in every aspect of our lives and for His mercy, protection, and provision during this research. Our unalloyed appreciation goes to all the staffs in the Department of Computer Science and Creative Technologies and the entire staff of the Faculty of Environment and Technology for their continuous supports, guidance, and invaluable suggestions throughout the duration of this research. God's wisdom and understanding to continue guiding students and colleagues will not depart from you all.

## REFERENCES

[1] Sanusi, B. A., Olabiyisi, S. O., Olowoye, A. O. and Olatunji, B. L. (2019). Software Defect Prediction System using Machine Learning based Algorithms. *Journal of Advances in Computational Intelligence Theory*, 1(3), 1–9. http://doi.org/10.5281/zenodo.35908 41

[2] X-machine. (2021). *In Wikipedia*. https://en.wikipedia.org/wiki/X-machine.

[3] Simons, A.J.H. and Leftticaru, R. (2020). A Verified and Optimized Stream X-Machine Testing Method, with Application to Cloud Service Certification. *Software Testing, Verification and Reliability*. 30(3): e1729. https://doi.org/10.1002/stvr.1729

[4] Kefalas, P., Stamatopoulou, I, Sakellariou, I. and Eleftherakis, G. (2008). Transforming Communicating X-Machines into P Systems. *Nat Comput*. 8:817–832 DOI 10.1007/s11047-008-9103-y

[5] Ogunshile E. (2011). A Machine with Class: A Framework for Object Generation, Intergration and Language Authentication (FROGILA). Ph.D. Thesis. The University of Sheffield. United Kingdom.

[6] Ogunshile, E.K.A. (2005). Automatic Generation of Java Code from Communicating X-machine Specifications. MSc (Eng) Thesis. Department of Computer Science. The University of Sheffield, England, United Kingdom.

[7] Leroy, X. (2009). Formal Verification of a Realistic Compiler. *Communications of the ACM*. Vol. 52. No. 7. Pages 107-115. DOI: 10.1145/1538788.1538814

[8] Mugridge, R., Utting, M. and Streader, D. (2011). Evolving Web-Based Test Automation into Agile Business Specifications. *Future Internet*. Vol. 3. Page 159-174.

[9] Akhtar, N. and Missen, M.M (2014). Contribution to the Formal Specification and Verification of a Multi-Agent Robotic System. *European Journal of Scientific Research*. ISSN 1450-216X / 1450-202X Vol.117 No.1. Page 35-55.

[10] Sharma, R. and Sangwan, M. (2016). To Improve Correctness, Quality and Reducing Testing Time. International Journal of Computer Science and Mobile Computing. (IJCSMC). Vol. 5. Issue 6. Page 211-217.

[11] Fonseca, P., Zhang, K., Wang, X. and Krishnamurthy, A. (2017). An Empirical Study on the Correctness of Formally Verified Distributed Systems. *EuroSys '17. ACM*. ISBN 978-1-4503-4938-3/17/04. DOI: http://dx.doi.org/10.1145/3064176.3064183.

[12] Chen, J., Donaldson, A. F., Zeller, A., and Zhang, H. (2017). Testing and Verification of Compilers. *Dagstuhl Reports*. Vol. 7. Issue 12. Page 50-65.

[13] Ogunshile, E.K.A. (2018). CompleX-Machine: An Automated Testing Tool using X-Machine Theory. International Journal of Computer and Systems Engineering 12 (3).

[14] Patterson, D. and Ahmed, A. (2019). The Next 700 Compiler Correctness Theorems (Function Pearl). Proc. ACM Program. Lang. 3, ICFP, Article 85. Page 1-29. https://doi.org/10.1145/3341689

[15] Koo, B., Bae, J., Kim, S., Park, K. and Kim, H. (2020). Test Case Generation Method for Increasing Software Reliability in Safety-Critical Embedded Systems. *Electronics* 2020, *9*(5), 797; https://doi.org/10.3390/electronics9050797.

[16] Olatunji, B. L., Olabiyisi, S. O., Oyeleye, C. A., Sanusi, B. A., Olowoye, A. O. and Ofem, O. A. (2020). Development of Software Defect Prediction System using Artificial Neural Network. *International Journal of Advances in Applied Sciences*. Vol. 9. No. 4. Page 284-293. ISSN:2252-8814. DOI: 10.11591/ijaas.v9.i4.pp284-293.

[17] Monteiro, F. R., Gadelha, M. R. and Cordeiro, L. C. (2021). Model Checking C++ Programs. Software Testing, Verification and Reliability. 32:e1793. Page 1-30. https://doi.org/10.1002/stvr.1793

[18] Elliott, G. (2004). Global Business Information Technology. An Integrated Systems Approach. Pearson Education. Page 87.

## AUTHORS

**Bashir Adewale Sanusi** is currently an Associate Lecturer in the Department of Computer Science and Creative Technologies, University of the West of England (UWE), Bristol, United Kingdom. He received B.Tech. and M.Tech. in Computer Engineering and Computer Science from Ladoke Akintola University (LAUTECH), Ogbomoso, Nigeria in 2015 and 2021 respectively. Also undergoing his Ph.D. degree in Computer Science from University of the West of England (UWE). He began his career in LAUTECH as a Graduate Assistant in 2018. He has published 6 peer-reviewed Journal articles and Conference paper.

**Dr Emmanuel Ogunshile**, Ph.D. is a Senior Lecturer in Computer Science and Chair Athena SWAN process at the University of the West of England (UWE), Bristoll, UK-conducting highly innovative Teaching, Research, Scholarship and Administration in Computer Science and Software Engineering. Previously, he graduated with an MSc(Eng) in Advanced Software Engineering (2005) following a BEng(Hons) in Software Engineering (2003) and a Ph.D. in the subject of Computer Science (2011) all at The University of Sheffield, England, UK.

**Mehmet Aydin** joined CSCT department at the end of January 2015 as Senior Lecturer in Computer Science. Prior to this post, I have worked in academic and research positions for various universities including University of Bedfordshire, London South Bank University and University of Aberdeen. I am editorial board member of a number of international peer-reviewed journals, and have been serving as committee member of various international conferences. I am also member of EPSRC Review College and fellow of Higher Education Academy.

**Stephen Olatunde Olabiyisi** is Professor of Computer Science in the Department of Computer Science, Ladoke Akintola University of Technology (LAUTECH), Ogbomoso, Nigeria. He received B.Tech., M.Tech and Ph.D. degrees in Applied Mathematics from LAUTECH, Nigeria, in 1999, 2001 and 2006 respectively. He also received M.Sc. degree in Computer Science from University of Ibadan, Ibadan, Nigeria in 2003. He began his career in LAUTECH as a Graduate Assistant in 2000 and rose through the ranks to become a Professor in 2013. He has supervised 51 Doctoral (Ph.D) students and 58 Master's students and published over 200 peer-reviewed Journal articles and Conference papers. He is currently the Director, LAUTECH ICT Center.

**Oyediran Mayowa Oyedepo** is a senior lecturer in the department of Computer Engineering in Ajayi Crowther University, Oyo town in Nigeria, he has a PhD in Computer Engineering from Ladoke Akintola University of Technology, Ogbomoso. Nigeria. He is specialized in Distributed Systems and Applications with focused on Cloud computing, fog computing, MANETs and the likes.