International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

Formal Methods as Specification and Verification Tools towards Stable Software Solutions

Adekola Olubukola D.[a]*, Adebayo Adewale O.[b]

[a,b]*Computer Science Department, Babcock University, Ilishan Remo, Ogun State, Nigeria*
[a]*Email: adexbukene@gmail.com*
[b]*Email: wale_adebayo@yahoo.com*

Abstract

Formal Methods could provide mathematical models for specifying and verifying designs- hardware or software. Early on, formal methods had more acceptance in hardware than software. Employing mathematical models in software to proof correctness or validate requirements reduces or eliminates errors at the early stages of development and also makes testing easier. Formal methods are powerful tools in introducing rigor that would enforce correctness in design specification and help build confidence in design. Indeed, formal method should be seriously considered in safety-critical systems where there is zero tolerance for failure. Formal methods have possibility of gaining magnitude because of the capability to formulate accurate solutions. This work proposes to look into the effects of mathematical models on software system designs from the perspective of formal methods. Trends, benefits, state of the art and future prospects of formal method are considered. Formal methods might require much in terms of implementation, skills and use but there is much benefit in terms of removing design ambiguity and inconsistency and at the same time improving correctness and accuracy.

Keywords: Ambiguity; Correctness; Mathematical models; Requirements; Specification; Verification.

1. Introduction

Method, in software engineering, simply means understanding a set of principles for selecting and applying a number of techniques and tools in order to analyze and synthesize (construct) an artefact [1]. A tool, in this context, is associated with specification and coding languages, model checking, theorem proving and test tools, among others.

---

* Corresponding author.

Techniques are abstract and concretization, proof, refinement and the like. Formal methods in software engineering connote a development that uses formal specification languages in various phases of development, domain specification, requirements and also in design. Formal Methods help in understanding a comprehensive set of methods, techniques and tools that have a formal background in mathematics. This means each specification language has both mathematical syntax and semantics, and a proof system and at the same time supporting refinement, model checking, proof and test [2]. Pioneers of computer science like C.A.R. Hoare and E.W. Dijkstra had developed the concepts of formal methods many decades ago but advances in the development of software tools recently promoted its popular use [3]. Perspectives on Formal Methods developed in the 60s and 70s had been in the area of hardware design verification [4]. The need for proof of correctness and prevention of persistent system failure gave more attention to the use of formal method in software design. It has been effective, also, in some safety-critical systems such as aircraft control, revenue reconciliation, expert systems and other related systems where there is zero-tolerance for failure [5]. Evolving programming languages and their types can be precisely described by grammar. The importance of formal grammatical representations of languages cannot be over-emphasized in the vast field of computer science, involving programming languages, compiler design, and natural language processing. The motivation behind implementation of formal method is to proof and ascertain correctness. In software or program design, correctness is about what can be proved mathematically. This is where formal methods belong [6]. An example of model-oriented methods is RAISE (Rigorous Approach to Industrial Software Engineering). RAISE was the formal method used at UNU-IIST (United Nations University International Institute for Software Technology) [7]. Formal methods are concerned with specification techniques in system design [8]. Special mathematical skills are required for experts to employ formal methods in standardizing designs. Quite a few researchers have the patience of undergoing the required rigor to acquire the essential knowledge before this could be used. Consequently, many system designers take less tedious approaches to specification and error checking even if the rigor in formal methods could possibly yield reliable solutions [9].

More so, formal methods are useful tools in reverse engineering to document findings about legacy systems and to discover and repair problems in the original specification [9]. Another area of software where formal methods have thrived is creation of protocols and algorithms.

Furthermore, formal method instruments such as Finite State Machine, Grammar, Syntax analysis tools, and so on, have helped greatly in verifying the structures of a system design which helps in eliminating potential problems right from the design and analysis phases. The merits of formal methods are that there is high propensity of building a usable and reliable product at first try and consequently reducing time for testing. It must also be noted that formal specification and methods are not alternative measure to testing but rather complementary.

As a matter of relevance, formal specifications, at top level of abstraction, are important instruments in software and design reuse [10]. Reuse measures, in their various forms are vital to speed, accuracy and productivity during software development. In today's world where speed and quality are essential, re-inventing the wheel becomes less fashionable, then reusability assumes a key concept in software design. To a great extent, the concept of software reuse has helped developers in meeting up with the market demands [11]. As it is, the trend

of development in today's design owes much to reuse. Reuse research areas such as product line engineering, component based development, service oriented systems, aspects and so on, therefore, are receiving much research attention. Invariably, with formal method supporting reuse, it has a tendency of gaining long time relevance.

In addition, model checking is one of the strongest formal methods which, in principle, simply generates every possible state of a program and checks if the correctness specifications are certified in each state [3].

The extent to which formal methods are relevant in system design cannot be over-emphasized and more research is ongoing in this direction.

2. Statement of Problem

Unlike hardware products, software has experience instability, especially at the early years, because of the negligence in crucial approach or rigor that can strengthen the products. As it has been well explored in hardware, formal methods could provide mathematical models for specifying and verifying software designs. This could serve as proof correctness or requirement validation in order to reduce or eliminate errors which could also provide leverage for safety critical systems. Furthermore, there is really a need for constant application of formal methods to complement testing.

3. Objectives

The main aim of this work is to address the potency of formal methods in improving the stability of software products.

The specific objectives are to:

i. Conduct extensive study on the applications and benefits of formal methods
ii. Evaluate the potency of formal methods in software and also in safety critical systems

4. Methodology

These studies embody case studies, systematic literature reviews and surveys. Important requirements were identified in related papers. The relevant documents obtained were qualitatively analyzed for convergence, and relevant details were extracted using inductive approach. This work also explored the concepts of formal methods, early achievements and the potency of its application in software in general.

5. Literature Review

5.1. Formal methods early impacts

Findings revealed that the United Kingdom Ministry of Defence once required the use of formal methods via code verification for special types of safety critical software [5]. The outcome of those applications recorded notable performance. Formal verification methods maintain huge safety integrity levels and procedures.

National Aeronautics and Space Administration (NASA) stated that the body implemented and incorporated formal methods as a crucial process in safety critical parts of flight software [12].

Formal methods are also gaining ground as verification and specification tools in standardizing software solutions based on legal factor. The fact that the developers realized that software failure can cause them to face tribunal make them incorporate methods such as formal method to standardize their design which could minimize or eliminate possibility of error and keep them away from the law court [9].

Formal methods should be applied at the requirements phase of development long before the thought of code implementation phase. That is, putting the formal method rigor on requirements specification goes a long way in preventive measure against error and in reducing time spent testing a system or software design. Results of requirement definition directly get better. Moreover, software experts own good reason to practice and use formal methods for the promising safety that its technicality provides.

5.2. Formal methods and varying requirements

In point of fact, Formal methods are not designed to completely replace testing but rather play a complementary role to testing [13]. It is meant to make testing quick and easy because the established mathematical model must have proven various aspect of the system. Formal method could help eliminate or reduce many test cases, and also improve the quality of specifications [14].

Table 1 presents some requirement types and corresponding formal methods that have been applied in addressing them.

Table 1: Some Requirement Types and Applicable Formal Methods [15]

s/n	Requirement Type	Applicable Formal Method
1	Functional Requirements	Deductive Verification
2	Communication and Protocols	Model Checking
3	Real-Time Requirements	Static Analysis
4	Memory Use	Run-Time Checks
5	Security	Run-Time Checks

The future trends of formal method could be multifaceted. Notably, one can mention the integration of formal method tools into (Computer-Aided Software Engineering) (CASE) tools, integration of formal methods into Software development process, combining test and formal verification, combining automatic methods with theorem provers, combining static analysis of programs with automatic methods and with theorem provers, and so on.

6. Effect of Formal Method on Post-Delivery System Behaviour

A system's behaviour and its effect after delivery are also important to the user and the designer. Post delivery time is the time a system is expected to serve the users/customers based on the specific purposes of the design. Many systems begin to fail as soon as they are deployed as a result of undiscovered errors during the development phases. The loopholes are exposed as soon as they are in their real environment where the users make use of them as naturally as possible. The errors here were consequent upon those that were not discovered during their unit or system testing phases of the system development life cycle. This happens because sometimes neither the developer (who does the requirement specifications) nor the customer (who narrates the system requirement) can visualize all the after effects of the use of the new software. Application of formal methods in specification and verification of system design can go a long way reducing the frequency of system failure after delivery [9]. This will help to extend the imaginations of both the developers and the users during requirement gathering, analysis, design, implementation and testing. This in itself makes formal method a substantial future for software development and indeed a way forward towards standardizing software products.

Formal methods are useful tools in the specification of systems during development and also at reverse engineering. It is essential that a sound understanding of the system and the process is gained by system designers before formal methods are applied. Notably, formal method works well in eliminating ambiguity from specification and design. Some form of formal training and understanding of mathematical models are required for developers to implement formal methods.

7. Why Formal Methods in Software Design

Over the years, product failure has been a persistent occurrence in the software industry [16]. This inherent failure goes with whooping loss, in terms of cost and time, and directly and negatively affects stakeholders' financial base and the economy at large. This leads to system rejection at various instances. There is, therefore, an increasing need for methods or innovations that can assure the success of software projects. For large and complex projects, an ad hoc approach has proven inadequate. The lack of formalization in key places makes software engineers responsible for many teething problems in the field. Techniques that influences precision and cross-checking are paramount and this is essentially the reason for formal methods.

The need to manufacture standardized but quality software cannot be over-emphasized and formal methods hold much promise in this regard. Software development is a vital part of our society today because barely everything is either automated or simulated. People will have to relate with software solutions or applications in their everyday life, from paying for food items at a grocery store to making a reservation, and so on. This is why

techniques for verifying the correctness of software solution would have a lasting relevance both now and in the future.

8. General goal of formal methods

Software development life cycle model can be summarized into the following phases: Requirements analysis (to break down user needs), Requirement specification (to describe concisely what the software will do), System design (to determine how to realize the software), Implementation (to construct the procedures, algorithms and programs), Verification (to check if the programs design meet the specification), and Maintenance (post-delivery upkeep and corrections on the product) [17]. The task of formal methods is to introduce certified mathematical rigor into all the phases of software development life cycle. This goes a long way addressing critical issues, providing standard means to record various assumptions and decisions, and forming a basis for consistency among many related activities.

8.1. Formal method concepts

The following are some notable formal method specification methods as well as the illustrations of their applicability (see Figure 1).

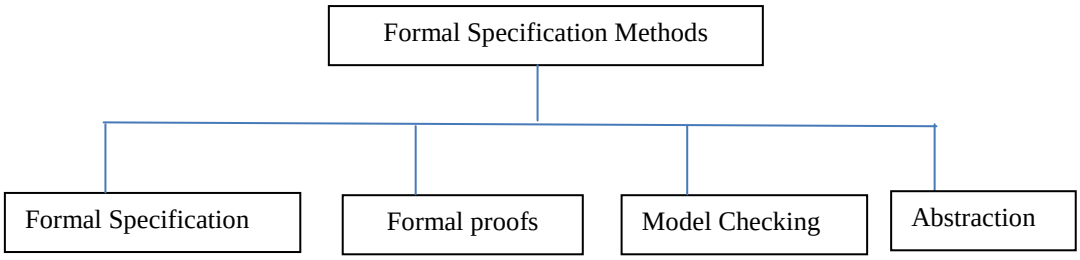

Figure1: Applications of Formal Specification Methods [18]

Formal specification involves translation of a non-mathematical description of a system into a formal specification language. Majorly, the goals are to overcome imprecision, ambiguity, and prolonged testing. **Formal Proofs** provides convincing arguments to validate some properties of system description. It connotes formulating series of steps and simple rules to formalize informal description which helps in removing ambiguity and subjectivity. **Model Checking** is about determining if the generated finite state machine model satisfies requirements expressed as formulas in a given logic. Here, all reachable paths in a computational tree derived from the state machine model are explored. **Abstractions** are used to simplify a system and to focus on general major features, which prevent undue concentration to design details especially when such details could distract or are not yet required [18].

8.2. Formal methods notable tools

Complex computation problems are simplified and are channelled to follow some series of formalized procedure that lead to solution phase(s) [2]. A tool like finite state machine for instance helps make complex

computational process easy to design, read and interpret. It is a mathematical model used to specify a system, a design, an algorithm or a solution. This will eventually consist of series of step by step simple procedure that will result into a solution to essentially large or complex problem. Regular expression, termed rules about rules, defines acceptable strings of letters. Basically, four rules connote the description of a regular expression namely: Alphabet rule, concatenation rule, alternative rule and repetition rule. Regular expression passes for a larger universe of possible formal systems. Conceptually, regular expression is supposedly not a sequence of steps but a description of a desired result. Formal method also presents grammar, which is very useful in programming language design because of the capability to create more efficient parse [19].

8.3. General benefits of formal specification methods

The following are the benefits attributed to formal methods and their applications:

a) Higher level of mathematical rigour promotes system builders' better knowledge of the system
b) Bugs that could subtly hide in traditional methods of specification are quickly exposed
c) Timely identification of defects at the early stage of system cycle
d) Gives room for formal proofs which can establish fundamental system properties
e) Provision of repeatable analytical procedure to authenticate theory.
f) Abstract formal view promotes understanding of what the system is to do without the need to bother about how it will be done.
g) Model checking activities and formal language for computer, software and software design are preventive measures toward bug or error management in system design.
h) Wrong requirement naturally translates to a wrong system. Therefore, it is highly constructive to use automated tools to check various aspects of requirement and design.
i) Model checking helps check if the architectural design and specification meets desired properties
j) With formal methods, burden at testing phase is minimized
k) Building formal language at requirement levels help remove assumptions and ambiguity

9. Results and Conclusion

Formal methods use mathematical models for analysis and verification at any phase of system development life cycle. Formal methods help both developer and client fine-tune and establish their system requirement more accurately. Formal methods go a long way in helping to detect errors at the early stages of system development especially at requirement level. The success of a design hinges heavily on getting requirement right. It is also worth noting that formal method is not a substitute measure to testing but rather complementary; it does not take the place of testing (which is an integral part of software engineering) but rather make testing as less costly as possible. Formal methods can tremendously improve quality assurance when carefully applied in system design. Formal methods gain more audience in security-inclined, reliability-driven and safety-critical applications. Moreover, it improves reuse, documentation, test case generation and maintenance. Furthermore, formal method has notable success in hardware domain but quite under-explored in software. Software community needs to harness the potent benefits in formal methods. Software engineers would have to acquire the knowledge needed

to be able to apply the mathematical rigor involved in using formal methods rather than choosing alternative verification measures. Tools like model and proof checkers will soon become adaptable and commonplace like compiler software. Also, it would represent a smart method for building system with minimized error, reduced cost and very high quality. More research in new specification language and new verification techniques are required. The place of further training for system developer to make this task easy cannot be overemphasized. Formal methods well explored should promote precision, accuracy and testability of software.

10. Limitations of the study

The limitation here is that the work does not make comparison in terms of the potency of application of formal methods in hardware vis-a-vis software. It also does not specify where this is better and easier to apply between the hardware and software industries.

References

[1] D. Bjørner. Software engineering 1: Abstraction and modelling. Theoretical Computer Science, EATCS Series. Germany: Springer-Verlag Berlin Heidelberd, 2006.

[2] D. Bjorner. A Survey of Formal Methods in Software Engineering; DTU Informatics, Denmark, Univ. of Macau, APSEC 2012, Hong Kong.

[3] M. Ben-Ari. Principles of the spin model checker. London: Springer-Verlag, 2008.

[4] D. Craigen and S. Gerhart. Formal methods reality check: Industrial Usage. IEEE transactions on software engineering, Vol. 21(2), 1995, pp. 90-98.

[5] S. King, J. Hammond, R. Chapman and A. Pryor. Is proof more cost-effective than testing? IEEE Transactions on Software Engineering, Vol. 26(8), 2000, pp. 675-685.

[6] G. J. Holzmann. The spin model checker: Primer and reference manual. Canada: Pearson Education, Addison-Wesley, 2004.

[7] G. Chris and P. Juan. Model checking RAISE specifications. The United Nations University, International Institute for Software Technology, Macao, China. 2006, UNU-IIST Report No. 331.

[8] S. Gerardo. Motivation on Formal Methods Formalisms for Specification and Verification Summary. Department of Informatics, University of Oslo, 2009.

[9] D. C. Stidolph and J. Whitehead. Managerial Issues for the Consideration and Use of Formal Methods. In: Araki K., Gnesi S., Mandrioli D. (eds.). FME 2003: Formal Methods. Lecture Notes in Computer Science, Vol 2805, Springer, Berlin, Heidelberg.

[10] J. P. Bowen. Ten Commandments of Formal Methods. IEEE Computer, Vol. 28(4), 1995, pp. 56-63.

[11] O. D. Adekola and O. Awodele. Enhancing Software Development through Software Product Line: Developing Product Family rather than Individual Products. International Journal of Advanced Studies in Computer Science and Engineering, Vol. 3(1), 2014, pp. 35-39.

[12] R. W. Butler, J. L. Caldwell, V. A. Carreno, C. M. Holloway, and P. S. Miner. NASA Langley's Research and Technology-Transfer Program In Formal Methods. Assessment Technology Branch, NASA Langley Research Center, Hampton, Virginia: Vigyan, 1995.

[13] M. Singh. Formal methods: A Complementary Support for Testing. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3(2), 2013, pp. 320-322.

[14] A. A. Saifan and W. B. Mustafa. Using Formal Methods for Test Case Generation According to Transition-Based Coverage Criteria. Jordanian Journal of Computers and Information Technology, Vol. 1(1), 2015, pp. 15-30.

[15] B. Beckert and R. Hähnle. Reasoning and Verification: State of the Art and Current Trends. IEEE Intelligent Systems, http://www.computer.org/intelligent, 2014, pp. 20-29.

[16] P. Dorsey. Top 10 Reasons Why Systems Projects Fail. NY: Dulcian, 2000.

[17] I. Sommerville. Software engineering (9th ed.). New York: Addison Wesley, 2011.

[18] A. E. Haxthausen. An Introduction to Formal Methods for the Development of Safety-critical Applications. DTU Informatics Technical University of Denmark, 2010, homepage.cs.uiowa.edu/~tinelli/classes/181/Fall15/Papers/Haxt10.pdf.

[19] J. E. Hopcroft, R. Motwani and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation (3rd ed.). Boston, USA: Addison-Wesley, 2006.