Volume 4, No. 5, May 2013
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

FORMAL METHODS: BENEFITS, CHALLENGES AND FUTURE DIRECTION

Mona Batra[1], Amit Malik[2], Dr. Meenu Dave[3]
[1] M. Tech. Scholar
Department of Computer Science, Jagan Nath University, Jaipur, India
monabatra89@gmail.com
[2]Sr. Analyst, HCL Technologies Ltd. , Noida-201304, India
malikamit1@gmail.com
[3]Assistant Professor
Department of Computer Science, Jagan Nath University, Jaipur, India
meenu.dave@jagannathuniversity.org

Abstract: There is an increasing demand of current information systems to incorporate the use of a higher degree of formalism in the development process. Formal Methods consist of a set of tools and techniques based on mathematical model and formal logic that are used to specify and verify requirements and designs for hardware and software systems. This paper presents a detailed analysis of formal methods along with their goals and benefits followed by limitations. This research work is aimed to help the software engineers to identify the use of formal methods at different stages of software development, with special reference to the requirements phase.

Keywords- Formal Methods, Requirements Engineering, Formal Specification, Feasibility Analysis etc.

INTRODUCTION

In today's commercial environment, the primary measure of success of software projects is the extent to which a software system fulfills the purpose, which it is intended for. However, an increasingly competitive market usually demands higher quality, shorter turnaround cycles, and cheaper software. Many IT companies face the difficulty of releasing products of quality on time and within the limits of approved budget. The number of errors found during development strongly affects the software metrics mentioned above. If the problem is identified earlier during program development, then it helps in reducing the project budget [1].

When an error that was introduced in the requirements phase is found during testing, software engineers must fix the incorrect requirements, check all of the ramifications through design and implementation and finally retest the product. In order to build a secure software product and overcome the problem of overrun of budget (which occurs due to errors in requirement specifications), cost-effective methods are required that address the major risks and that provide tangible evidence of trustworthiness.

Formal methods are the solution to the above stated problems. Formal methods are a particular kind of mathematical techniques meant for the specification, development and verification of software and hardware systems. The representation used in formal methods is called a formal specification language. [2].

Formal methods consist of writing formal descriptions, analyzing those descriptions and in some cases producing new descriptions. They can be applied in different phases of development process. They are even becoming integral components of standards. According to Rushby [3], the use of mathematics in design and construction to ensure product quality is common practice in established engineering disciplines, such as bridge or aircraft building, and even computer (hardware) construction, where one applies mathematically expressed physical and other natural laws to model a problem that deals with the behavior of concrete systems in the physical world.

This paper describes various aspects of formal methods in requirements engineering. Formal specification language, different formal specification styles and types of formal methods are described in Section 2. In Section 3, goals of formal methods are explained whereas in Section 4 benefits of formal methods are discussed in detail. In Section 5, limitations are presented, and the issues in Section 6. 'Conclusion and Future Work' is reported in Section 7.

TYPES OF FORMAL METHODS

Formal methods are mathematics based languages, techniques and tools that can be applied to any part of the program life-cycle. By providing the feature of abstraction and unambiguous description mechanisms, formal methods facilitate the development of the critical systems. The representation used in formal methods is called a formal specification language. The formal specification languages are based on set theory and first order predicate calculus. The language has a formal semantics that can be used to express specifications in a clear and unequivocal manner. Formal methods can be classified using two ways. Firstly, according to formal specification styles, and secondly, according to software development life-cycle perspective [4].

TYPES OF FORMAL SPECIFICATION STYLES

The Formal Specification Styles are specified as follows:

Model Based Languages:

Model based languages are a way to write a specification. To specify the behavior of the system, model based languages construct a mathematical model of the system. The model consists of an underlying state (data) and a collection of operations on that state [5]. The state model is constructed with the help of mathematical entities such as relations, sets, sequences and functions. Operations of a system are specified by defining how they affect the state of the system model. Operations are also described by the predicates given in terms of pre and post conditions. The most widely used notations for developing model based languages are Vienna Development Method (VDM), Zed (Z) and B.

Algebraic Specification

Algebraic specification is a technique, used to specify the system behavior by using methods derived from abstract algebra. Algebraic approach was originally designed for the definition of abstract data types and interface. The most widely used notations for developing algebraic specification languages are LARCH, ASL and OBJ.

Process Oriented:

Process oriented formal specification language is basically used to describe concurrent system by building a specific implicit model. In these languages processes are denoted by expressions and are built up with the help of elementary expressions. Elementary expressions describe simple processes along with the operations, which combine processes to yield more complex processes. The most widely used process oriented language is Communicating Sequential Processes (CSP).

FORMAL LANGUAGES IN SDLC

In SDLC, formal languages are used in two phases: requirements and testing.

Specification (Requirements Analysis Phase):

Specification is the process of describing a system behavior and its desired properties. Formal specification languages describe system properties that might include functional behavior, timing behavior, performance characteristics and internal structure, etc. [6].

Z, VDM and Larch are used for specifying the behavior of sequential systems while other formal methods such as CSP, CCS, State charts, Temporal Logic, Lamport and I/O automata, focus on specifying the behavior of concurrent systems [7]. RAISE is used for handling rich state spaces and LOTOS is one of the languages for handling complexity due to concurrency.

Verification (Testing Phase):

Verification is the process to prove or disprove the correctness of a system with respect to the formal specification or property. For the verification of the code, there are two important forms: Model Checking and Theorem proving [8].

a. In model checking, a finite state model of the system is build and its state space is mechanically investigated. Two well-known and equivalent model checkers are NuSMV and SPIN.
b. Theorem proving is another approach for verification of a specification or checking the correctness of a program. A model of the system is described in a mathematical language and desired properties of the model can be proven by a theorem prover. It is mechanization of a **l**ogical proof. The specification to be checked by a theorem prover is written in a mathematical notation. Z (pronounced 'Zed') is its well-known example.

GOALS OF FORMAL METHODS

Formal methods can be applied at different stages of software development life cycle. On the basis of the details of the formal methods, some goals may be listed as follows:

a. Formal methods support in the creation of specifications that describe the true requirements of the user, which are not usually identical to the stated requirements. This can be achieved using formal methods because of the unambiguity of the formal specifications and the possibility to prove certain properties about it.
b. Formal methods ensure that the implementation of a particular software as well as hardware product should satisfy the requirements specification.
c. Formal methods are basically concerned for development and maintenance of security critical reliable systems on time and within budget. It increases trustworthiness of the system in the sense that the system developed is not just correct but known to be correct. Formal methods act as evidence which ensures that the system indeed satisfies the demand of security, reliability and correctness.

BENEFITS OF FORMAL METHODS

The early activities in the software development lifecycle i.e. requirements analysis and specification, is the most important. According to one of the Standish Chaos report [9], half of all project failures occur due to poor requirements specification. The most effective use of formal methods is at these early stages. It is effectual to write a specification formally rather than writing an informal specification and then translating it. To detect inconsistency and incompleteness, it is efficient to analyze the formal specification as early as possible [10]. Along with the benefits discussed above, there are various other benefits which are discussed as below:

a. ***Measure of correctness:*** The use of formal methods provides a measure of the correctness of a system, as opposed to the current process quality measures.
b. ***Early defect detection:*** Formal Methods can be applied to the earliest design artifacts, thereby leading to earlier detection and elimination of design defects.
c. ***Guarantees of correctness:*** Formal analysis tools such as model checkers consider all possible execution paths through the system. If there is any possibility of a fault/error, a model checker will find it. In a multi-threaded system where concurrency is an issue, formal analysis can explore all possible interleaving and event orderings. This level of coverage is impossible to achieve through testing.
d. ***Error Prone:*** Formal description forces the writer to ask all sorts of questions that would otherwise be postponed until coding. This helps to reduce the errors

that occur during or after coding. Formal methods have the property of completeness, i.e. it covers all aspects of the system.

e. ***Abstraction:*** If the working of software or hardware product is simple, then one can write the code straight away, but in the majority of systems the code is far too big, which generally needed the detailed description of the system. A formal specification, on the other hand, is a description that is abstract, precise and in some senses complete. The abstraction allows a human reader to understand the big picture of the software product easily.

f. ***Rigorous Analysis:*** The formality of the description allows us to carry out rigorous analysis. Formal descriptions are generally written from different points of view, by which one can determine important properties such as satisfaction of high level requirements or correctness of a proposed design.

g. ***Trustworthy****:* Formal methods provide the kind of evidence that is needed in heavily regulated industries such as aviation. They demonstrate and provide concrete reasons for the trust in the product.

h. ***Effective Test Cases:*** From formal specification, we can systematically derive effective test cases directly from the specification. It's a cost effective way to generate test cases.

LIMITATIONS OF FORMAL METHODS

Formal methods play an important role in software development lifecycle. Yet, these methods have some limitations. These shortcomings limit the effectiveness of the formal methods for software products. Some of the limitations of formal methods are listed below:

a. ***Correctness of Specifications:*** Generally, actual user requirements might be different from what the user states, and will usually vary with time. While using formal methods, there is no way to guarantee correctness and completeness of a specification with respect to the user's informal requirements. However; various approaches exist in literature to reduce the probability of incorrect specifications, but the starting point of all approaches is necessarily informal. One can never be sure to have gathered all user requirements correctly.

b. ***Correctness of Implementation:*** It is very difficult to identify whether or not a given program satisfies the given specifications. For example, when using one of the verification checking approach such as Hoare logic, one needs to identify the loop invariants, which is not possible automatically. As a result, it is often impossible to prove the correctness of an existing program that has not been written with the correctness proof in mind. Correctness proofs are only feasible if programming and proof go simultaneously.

c. ***Correctness of Proofs:*** Correctness proofs play an important part in formal methods. Correctness proofs increase the probabilities that the program is correct. It is generally impossible to ensure about the correctness of specification as well as implementation. The main problem in the proofs lies in the creation of the proofs. Sometimes, there is a possibility that proof of correctness might fail. The possible reasons why the proof of correctness of an implementation with respect to its specification might fail [11] are:

a) The program is incorrect and needs to be modified.
b) The program is correct, but the correctness proof has not been found yet.
c) The program is correct, but there is no correctness proof.

a. ***Dealing with complex language features:*** Formal definitions of semantics of most of the important language constructs and software system components are either not available or too complex to be useful. For proving the properties of programs, these constructs or components would actually be required. Some of them are complex data structures, pointers, human-computer interface (HCI) and error messages etc. Generally, more than half the code of any real production system consists of HCI and error messages.

b. ***The Technical Environment:*** A formal description of the program should contain a description that a program is to work in coordination with hardware and under the specification of operating system in order to prove the correctness of the program. Generally such a formal description is often not available for the kind of technical environment used in industrial software development. The problem is worsened by the fact that such a formal description has to take a very specific form depending on the formal method used (for example as a theory to be used in a Theorem Prover). This applies to both the development environment and the production environment. As for the development environment, a formal definition of the programming language used and its semantics as implemented in the compiler are needed.
Additional complications are introduced by the following aspects of the environment:

a) Rounding errors in computations with floating point numbers. These are the reason why formal methods are not usually applied to numerical algorithms.
b) Size limitations.

ISSUES NOT ADDRESSED BY FORMAL METHODS

There are some of the issues that are not addressed by formal methods. These are discussed below:

a. ***Creativity:*** Formal methods are descriptive and analytical in nature. They are not considered to be creative. In reality, there are only formal ways of describing and analyzing designs. There is no such thing as a formal design process. In order to develop a real system we must combine formal methods with other approaches.

b. ***Software product quality:*** Formal methods deal with the software itself and its documentation. Other important components of software products such as training, customer support, maintenance or installation of the software, have to be dealt with separately. These components and their quality together form a quality product. Formal methods do not contribute in software product quality. As a result, most of the successful providers of software products have to put a lot of effort into addressing all the relevant aspects of a software product.

c. ***Software systems and their social and ecological environment**:* Software system normally takes inputs from external environment. These inputs may not be predictable. This obvious ignored issue usually creates the problem of developing `correct' specifications and deciding what behavior is correct. Formal methods can contribute nothing towards this aspect of software system.

FUTURE RESEARCH DIRECTIONS

FM (Formal Methods) is a very active research area with a wide variety of methods and mathematical models. In current scenario, there is not available any one method that fulfills all the security related needs of building a secure formal specification. Researchers and practitioners are continuously working in this area and thereby gaining the benefits of using formal methods. Moreover, future work needs to be done in any of the following research areas as represented in Figure 7.1).

a. Work may be initiated to develop a formal method that combines various benefits of other methods that focus in building secure formal specification.
b. Work may be done to reduce the cost of using formal methods in different phases of SDLC.
c. Identifying and addressing various formal specification verification tools.
d. Further research is required to make use of abstraction in combining multiple mathematical theories.
e. It is needed to scale up the notations of formal methods and the tool support to make it easy to use.
f. Work may be commenced on optimizing methods and tools for finding errors so that correctness to the system is identified.
g. Research may be carried out to amortize the cost of a method or tool over many uses. It should be possible to derive benefits from a single specification at several points in a program's life cycle: in design analysis, code optimization and testing.
h. Work may be initiated on developing a tool that helps in understanding how to compose methods, specifications, models, theories, and proofs.
i. A new mathematical model may also be developed for checking the completeness and logical consistency of requirements specification

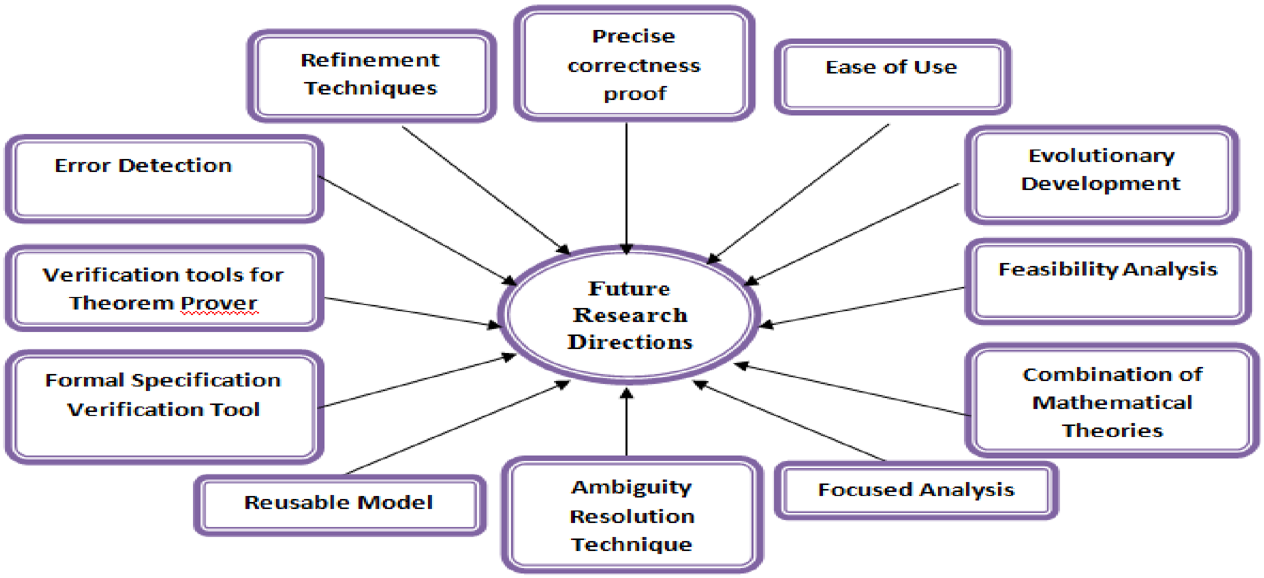
Refinement Techniques
Precise correctness proof
Ease of Use
Error Detection
Evolutionary Development
Verification tools for Theorem Prover
Future Research Directions
Feasibility Analysis
Formal Specification Verification Tool
Combination of Mathematical Theories
Reusable Model
Ambiguity Resolution Technique
Focused Analysis

Figure 7.1 Future Research Direction

CONCLUSION

This paper has presented different aspects of formal methods. The major defects arise in software development due to poor requirements analysis. Furthermore, formal methods are only part of the solution to the problem related to requirement analysis and success depends crucially on integrating them into a larger process. This paper helps the researcher/s and developers to understand the potential usefulness of formal methods along with challenges usually faced in making formal methods practical. Significant need in software development is needed to make all the methods to be more specific for the requirements phase because requirements are basic building block on which the entire software can be built. This work motivates software engineers to incorporate security in requirement phase so that product quality can be achieved.

ACKNOWLEDGEMENT

The authors thank Mr. Manish Gupta, Chancellor, Jagan Nath University, Mr. Deepak Gupta, Vice-Chairman, Jagan Nath Gupta Memorial Education Society, Prof. V.K. Agarwal, Vice Chancellor, Jagan Nath University, and Prof. Y.S. Shishodia, Pro-Vice Chancellor, Jagan Nath University, Jaipur, for providing encouragement and support for this research work.

REFERENCES

[1]. Boehm B. W.: Software Engineering Economics. Prentice Hall, 1981.

[2]. Pressman Roger S: "Software Engineering"- A Practitioner's Approach", McGraw Hill, 5th edition. 2000.

[3]. Rushby John: Formal Methods and the Certi_cation of Critical Systems. Tech. Rep. SRI-CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park, CA, Dec. 1993. Also issued under the title "Formal Methods and Digital Systems Validation for Airborne Systems" as NASA Contractor Report 4551, December 1993.

[4]. Mona Batra, S.K Pandey: Formal methods in requirement engineering. International Journal of Computer Applications , pp- 7-14, Volume 70–No.13

http://research.ijcaonline.org/volume70/number13/pxc3888017.pdf

[5]. McGibbon Thomas: An Analysis of Two Formal Methods: VDM and Z. ITT Industries - Systems Division.

[6]. Woodcock Jim, Larsen Peter Gorm, Bicarregui Juan and Fitzgerald John: Formal Methods: Practice and Experience, ACM Computing Surveys (CSUR), Volume 41 Issue 4, October 2009 Article No. 19. Retrieved on: March 17, 2013.

http://deployeprints.ecs.soton.ac.uk/161/2/fmsurvey%5B1%5D.pdf

[7]. Paul Ogilvie: Formal Methods in Requirements Engineering. Retrieved on: March, 17,2013

http://www.google.co.in/url?sa=t&rct=j&q=paul%20ogilvie%20formal%20methods%20in%20requirement%20engineering&source=web&cd=1&cad=rja&ved=0CC4QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.93.8000%26rep%3Drep1%26type%3Dpdf&ei=ispFUeXGBNCzrAfP6oHgDQ&usg=AFQjCNHuCyMppgBZ5cxA0QAOhfOIYZL8Lw&bvm=bv.43828540,d.bmk

[8]. Kneuper Ralf: Limits of Formal Methods, Formal Aspects of Computing (1997). Retrieved on : April, 20, 2013.

http://link.springer.com/content/pdf/10.1007%2FBF01211297.pdf#page-1

[9]. Standish Group, 1995, The Standish Group Chaos Report.

http://www.projectsmart.co.uk/docs/chaos_report pdf.

[10]. Hall Anthony: Realising the Benefits of Formal Methods. Retrieved on : April, 22, 2013.

http://www.anthonyhall.org/csi.pdf

[11]. Fuxman Ariel Damian: Formal Analysis of Early Requirements Specifications.

Short Bio Data for the Authors

Mona Batra is an Assistant professor in the Department of Computer Science, International Institute of Management, Engineering & Technology, Jaipur, India. She has completed her B. Tech from Rajasthan Technical University in 2011 and currently pursuing M. Tech (Computer Science) from Jagan Nath University, Jaipur. She has published various national and international papers on requirements engineering and security. Some of her representative published papers list is as follows: "Security in requirements phase of SDLC" published in IJCA, "Formal Methods in requirements phase of SDLC" published in IJCA, "Proposed model for requirements engineering and risk analysis" published in the national conference (AICC). Her research area includes: Vulnerability Assessment, Formal Methods and Requirements Engineering etc. Currently, she is working in the area of Formal Methods in Requirements Engineering.

Amit Malik is Senior Analyst in HCL Technologies Ltd, Noida, India. He has completed his MCA from Rajasthan Technical University in 2009. He has extensive experience in analysis, design and development of applications. He has worked in product based and client based software applications in various domains.

Dr. Meenu Dave, M.Tech., Ph.D. (Computer Science) has taught Computer Science in different capacities at a number of Engineering Colleges and Institutes. At present, she is an Assistant Professor in the Department of Computer Science, Jagan Nath University, Jaipur. She has extensive experience in teaching Artificial Intelligence, Knowledge Management and Data Mining at the postgraduate level. She has also authored several research papers in the specified areas which have been published in leading journals.