IMPACT: International Journal of Research in Engineering and Technology
ISSN (P): 2347–4599; ISSN (E): 2321–8843
Vol. 11, Issue 3, Mar 2023, 1–9

SOFTWARE REQUIREMENTS ENGINEERING THROUGH FORMAL METHODS

Seshadri Comandur Srinivasa

Research Scholar, Department of Instrumentation and Control, National Institute of Technology, Trichy, TamilNadu, India

Received: 02 Mar 2023 **Accepted:** 04 Mar 2023 **Published:** 08 Mar 2023

ABSTRACT

The goal of this work is to examine formal methods, their analogies, trends, environments, verification requirements, and their applicability to software engineering. The term "Formal Methods" describes approaches and tools that are rigorously mathematically used in the specification, design, and verification of software. In order to hasten the creation of requirement-based test cases, it also suggests a framework for a solution that uses formal methods. In addition, this addresses potential applications for this technique in a typical software development life cycle (SDLC). Also given are a few recommendations for certain formal method techniques that can help with partial or full automation of the auto generation of requirement-based test procedures or test scripts.

KEYWORDS: *Formal Method, SDLC, Engineering, Software, Hardware*

1. INTRODUCTION

When compared to its other traditional engineering rivals, such as civil and mechanical engineering, software as a subject is only a little over three decades old, making it quite young and relatively less developed [1]. In addition, programmers are fallible people who make mistakes. Many proficient programmers spend up to half of their time finding and correcting mistakes that they and their team members made in the other half. Most programming languages and tools don't actually have error-proofing built in. It is not unexpected that software products frequently arrive behind schedule and with functionality that needs years to develop in order to satisfy the needs of the original client. It gets more and harder to track requirements as they inevitably evolve, as well as changes to the environment where software is used [2]. Even today, software engineering is still mostly done as an art rather than a science, which likely explains why, in contrast to its counterpart goods in traditional engineering, very few software products in the current market place are guaranteed for an extended period of time.

The modelling and mathematics used in software definition, design, and verification are referred to as "formal methods." The development of theories and technologies to support these tasks is the main focus. An engineer can use it to write specifications that are more thorough, consistent, and clear than those made using traditional or object-oriented techniques. Logic notation and set theory are used to construct a precise explanation of facts (requirements). The validity and consistency of this mathematical definition can subsequently be demonstrated by analysis. For the past ten years, Airbus has used SCADE to develop DO-178C Level A controllers, such as the Flight Control Secondary Computer and the Electric Load Management Unit, for the A340-500/600 series [3].The specification is intrinsically less confusing than informal means of expression because it is constructed using mathematical language. The techniques are "formal" in that they are accurate enough to be used with computers. With the use of these methods, we may create specifications and

models that, at different degrees of abstraction, define all or a portion of a system's behaviour and use them as input for an automated theorem prover. A report on where discrepancies are found may be the product of requirements engineering, which may have as its input a collection of incomplete specifications. A specification and a design step may be the input for design, and the output may be "Yes, the design step is consistent with the specification" or "No, and here's why not: ". A specification and a desired system behaviour attribute may be the inputs for verification, and the output may be "Yes, the property is a consequence of the specification." or "No, and here's why not:. According to a US Small Business Association report from 2002, the software industry itself was only worth $220 billion. Even today, software engineering is still mostly done as an art rather than a science, which likely explains why, in contrast to its counterpart goods in traditional engineering, very few software products in the current market place are guaranteed for an extended period of time [4].

As its name suggests, the Requirement Maturity Index (RMI) is a metric used to assess the degree of trust in the requirements definition procedure [8]. This is not a quality control measure, rather a quality assurance activity. We might increase trust in the maturity and completeness of the requirement in an objective manner by enforcing a few toll-gate checks during the requirement review process, including the model checker and theorem provers report (Formal techniques). Before a need is flowing down to the design phase, it may now be able to measure its maturity thanks to formal approaches. Even if the computed RMI is low, employing formal methods would clearly show the problems in the requirement document that need to be fixed (by producing a counter-example) to enhance this score objectively.

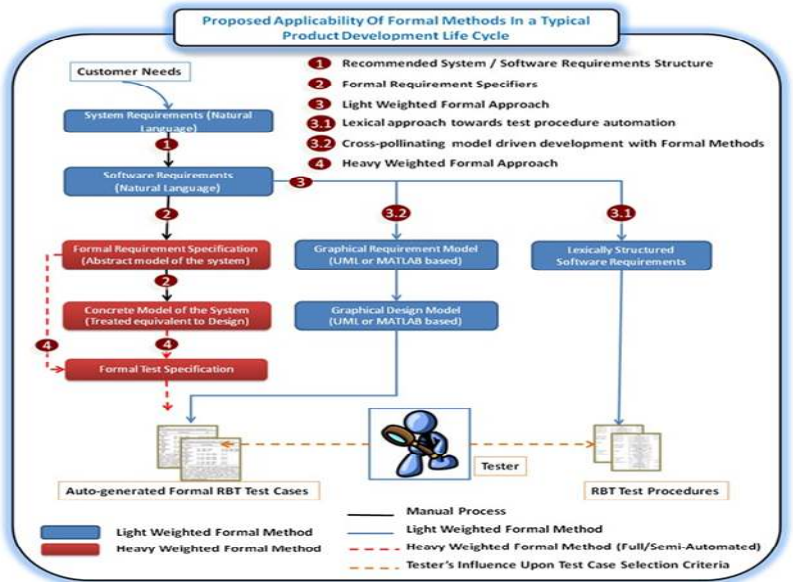


Figure 1: Recommended Structure.

2. LITERATURE REVIEW

2.1 Comparing it to Engineering Mathematics

Engineers in more conventional engineering fields (like civil or mechanical engineering) create mathematical models of their designs and utilise calculations to determine whether the design satisfies its requirements when placed in the context of a modelled environment. For the purposes of behaviour modelling, exploration, and debugging (/refutation), these mathematical models are utilised in the design loop. Also, these model artefacts will serve as reliable proof for the

certification of these goods. But first, it must be confirmed that the model accurately captures the design, that the design is executed appropriately, that the environment is accurately modelled, and that calculations are carried out without error [6].

Formal Logic is the name given to the same concept when it is applied to computational systems. In a nutshell, it is a subfield of computer science that deals with computational applications of applied mathematics. The models are logical systems that contain formal descriptions of all represented actions. Additionally, the majority of the calculations can be automated with this representation using automated deductions from a number of formal techniques, such as theorem provability, model checking, static analysis, etc.

2.2 Analogy with Simulation, Testing, and Other Formal Techniques

A model of the system that is specifically created for execution rather than analysis is also taken into account during simulation. Tests take into account the real thing. Both depart from formal techniques in that they focus on a small number of potential behaviours. Verification by extrapolation from partial tests is valid for continuous systems, but it is invalid for discrete systems, leaving us with no other option but to make statistical projections, which is very expensive. In addition, testing is typically utilised for debugging rather than verification in programmes.

Formal methods of tracking requirements are built on a mathematical approach to the specification, development, and verification of software and hardware systems. The use of widely accepted notation can be considered a formal approach, as can the full formality of theorem proving or automated deduction, the most advanced branch of automated reasoning at the moment. Automated reasoning (AR) is a technique for having computer software proves mathematical theorems [7].

2.3 Practical Obstacles

Although there has been some modest progress in using formal approaches with software, it is still not being fully utilised. A significant barrier is the widespread perception among software engineers that formal notations and formal analysis techniques are difficult to understand and apply. Additionally, formal approaches' scalability and cost-effectiveness are frequently seriously questioned by software developers. The absence of two aspects typical of hardware design environments in practical software development is another factor contributing to the modest influence of formal approaches. Initially, to specify their designs, hardware designers typically utilise one of a select few languages, such as Verilog or VHDL [8]. In contrast, software development rarely employs exact specification and design languages. Second, incorporating a formal technique, such as a model checker, into a hardware design process is comparatively easy because other tools, like as simulators and code synthesis tools, are already a standard part of the design process at many hardware organisations. In contrast, there are no standardised software development environments in the field of software development. Defining the requirements and compliance to meet the safety aspect of the systems often pose major challenges and are mandated by industry standards and certification agencies. The conference article titled “Control System Software Execution During Fault Detection” presented at 2022 6th International Conference on Intelligent Computing and Control Systems and published in IEEE discuss the safety aspect of the system and its protection [18]. The reliability aspect of the system is an important factor to be considered in requirements and define them in the early stage. The article titled “Framework for Data Management System to Assist Aircraft System Maintenance” discuss about the framework and defines the predictive system to improve the reliability of the system [19].

2.4 A Major Obstacle in Formal Methodologies

Each great challenge's main motivation for being created and announced is to enhance a particular field of science or engineering. A great challenge is an agreement by a sizeable portion of the scientific community to collaborate on an endeavour that has been determined to be worthwhile and doable by a group effort within an anticipated timeframe. In order to do this, Tony Hoare, Principal Researcher at Microsoft Corporation, revived an old grand challenge aimed at developing and applying a "Verification Compiler," which ensures a program's correctness before it is even performed [9]. In order to emphasise a dedication to the typical objectives and procedures of scientific research, this major challenge was covered at the conference titled "Verified Software, Theories, Tools and Experiments," sponsored by the International Federation of Information Processing (IFIP), held in Zurich, Switzerland, in October 2005 [10].

This is an excerpt from Tony Hoare's lecture, "Zero Defect Programming: The Impossible Dream," in which he discussed his two main goals for tackling this enormous task.

3. METHODOLOGY

An excerpt from John Rushby's talk, "Tutorial Introduction to Modern Formal Techniques," is shown in figure 2 [5]. As a result, these four separate planes can be used to conceptualise the applicability of formal methods:

- Interactive proofs of theorems
- Automatic Proofs of Theorems (based on abstraction)
- Model Validators
- Hidden Formal Techniques

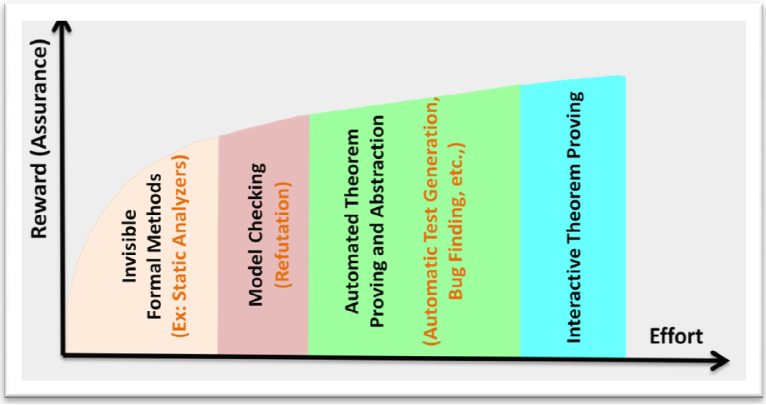


Figure 2: A Purview of Landscape.

Interactive Proofs of Theorems

A branch of computer science and mathematical logic called interactive theorem proving investigates strategies for creating formal proofs with the use of computers. This requires a proof aid of some kind, such as an interactive proof editor or other interface, via which a person can direct the search for proofs, the information for which is stored in a computer's memory and some of which are provided by the computer.

Interactive theorem provers need human input in the form of hints. Depending on the level of automation, the prover may

be reduced to little more than a proof checker, requiring the user to formally provide the evidence, or it may be possible for important proving activities to be carried out automatically. Although fully automatic systems have proved a number of interesting and challenging theorems, including those that have long confounded human mathematicians, interactive provers are employed for a variety of tasks. These accomplishments, nevertheless, are rare, and working on challenging issues typically calls for an experienced user [11].

Automatic Proofs of Theorems (Based on Abstraction)

Automated theorem proving, often known as automated deduction, is the process of having computer software prove mathematical theorems. The most developed area of automated reasoning at the moment is ATP (AR). The user must state the theorem, put it in a form that can be solved, and typically adjust certain parameters so that the theorem proving solves the problem in a fair amount of time after the computer has given it to them [12]. Example: Ices

Model Verification

Model checking in computer science is the process of automatically determining whether a system model conforms to a given specification given a model of the system. The specification typically includes safety requirements such the absence of deadlocks and other similar critical states that can result in the system crashing. These systems are either hardware or software systems.

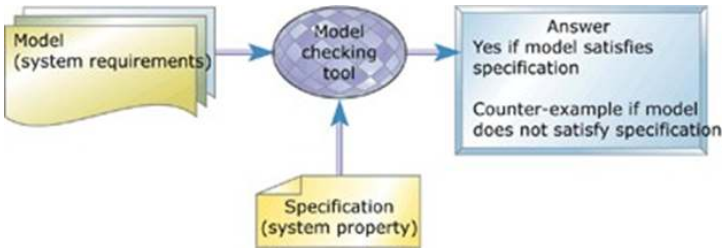


Figure 3: The Essential idea Behind Model Checking.

The most effective method that has emerged for verifying requirements is model checking. It is helpful to model check a downscaled instance before demonstrating the general case of a theorem [13]. But, incorporating model checking into a general case proof is a more intriguing combination. Because the search space is bounded and we are aware of its structure, model checking is both efficient and safe. Example tool: 1. SPIN 2. UPPAAL

Formal Invisible Techniques

Model As comparison to traditional (Interactive Theorem Provers) formal verification, checking for refutation and for verification (through automated abstraction) imposes a substantially lower adoption hurdle. The barrier is still present, though. Therefore, efforts are being made to remove formal methods from traditional tools and to offer a graded sequence of formal analysis technologies, starting with extended type checking and progressing through approximation and abstraction to model checking and theorem proving, in order to implicitly transfer these advantages to a wide range of end users [14].

4. RESULTS AND DISCUSSIONS

The study "Formal methods: Practice and Experience" by Jim Woodcock from the University of York, Peter Gorm Larsen from the Engineering College of Aarhus, Juan Bicarregui from the STFC Rutherford Appleton Laboratory, and John

Fitzgerald from Newcastle University describes the state of the art in the industrial use of formal methods with a focus on their rising use at the earlier stages of specification and design [11]. One of the most important surveys conducted in the past 20 years was this one. The excerpt from their published paper that follows offers a more recognisable and practical use of formal methods in the computational world.

A structured questionnaire was used between November 2007 and December 2008 to collect information on 62 industrial projects across a variety of industry segments, including transportation, defence, and consumer electronics that were known to have used formal techniques from published literature, mailing lists, and personal experience [15]. The projects that were surveyed originated in Europe, North America, South America, Australia, and Asia (in decreasing order).

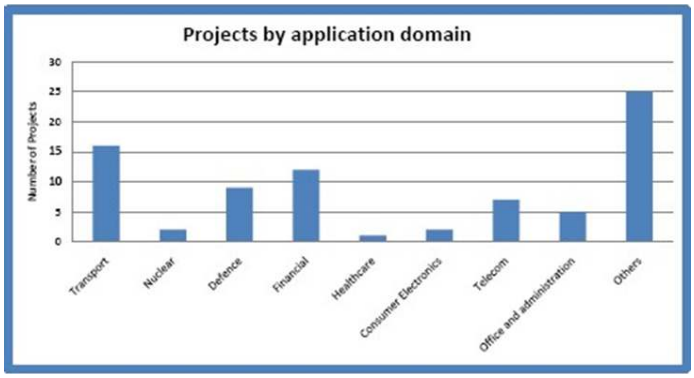


Figure 4: Application Domain.

The assessment took into account the wide range of formal verification techniques being used in practise.

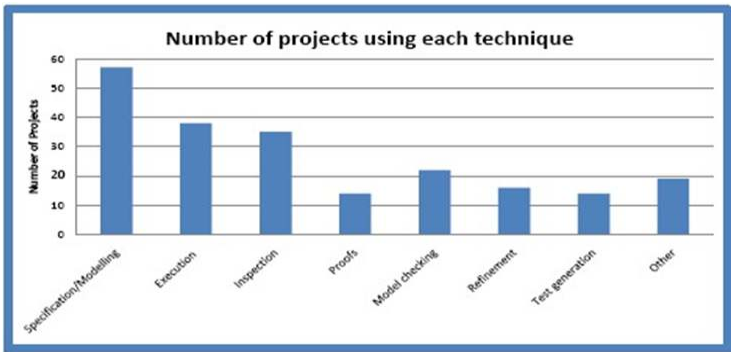
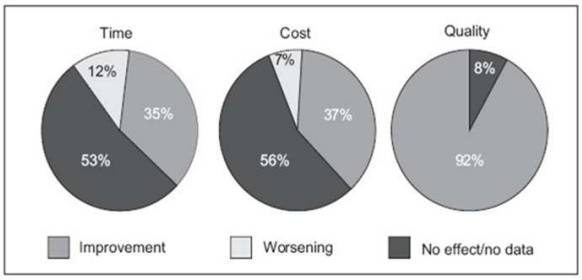


Figure 5: Formal Verification used Techniques.

The survey's undisputed findings indicate a significant improvement in the quality of the work product (92%), with no examples citing a decline in quality. Among the explanations given are:

- Early fault discovery (36%).
- Design enhancements (12%),
- Enhanced accuracy confidence (10%),
- Enhanced understanding (10%),
- Additional issues (4%).



Did the use of formal techniques have an effect on time, cost, and quality?

Figure 6: Result of the Survey.

Five times as many projects reported lower costs as higher costs out of the 44% of the projects surveyed that had data on the expenses associated with formal methodologies.

- The cost rise was largely caused by a lack of exact, comprehensive knowledge regarding the required, externally visible behaviour of the software product, among other factors. Applying formal specification and formal verification was reasonably simple once the needed behaviour was made obvious.
- Annotating the code with pre- and post-conditions was the only costly step in the code verification procedure. It was simple to demonstrate the relationship between the annotated code and the abstract specification of the desired behaviour once the annotated code was available [16]. During the second procedure, faulty annotations were fixed and additional annotations were added. The abstract specification and necessary security features barely changed throughout this procedure.

On the whole, the effect on how long the work took to complete was favourable. Instead of an increase in time, three times as many people reported a decrease [17]. Many replies stated that it was impossible to assess the impact on time spent, while a few did mention longer specification times, which may or may not have been offset by shorter system integration and testing times later on.

5. CONCLUSIONS

In contemporary software development practise, the execution and monitoring of tests are largely automated. But creating test cases has always been a labour-intensive manual process. The moment has never been more ripe and appropriate for using formal methods in industry-scale projects than it is now, as more and more certification standards are beginning to accept and appreciate the inherent potential of these techniques (like DO178C, for example). Automated test creation is a fascinating topic in this field since it can lower testing costs and possibly raise testing quality. Also, it is an "invisible" use of formal methods, which presents a significant potential to lower the industry's adoption barrier. Early results from the pilot projects are encouraging, and if they are used wisely, we can assume that the software costs will be reduced by at least 40–50% and that the time to market will be shortened by at least 20–30%.

6. REFERENCES

1. *Parnas, D. L. and J. Madey, "Functional Documentation for Computer Systems Engineering, Vol. " McMaster University, Hamilton, Ontario, Technical Report CRL 237, September 1991.*
2. *D Parnas, J Madey, Functional Documents for Computer Programs, Science of Computer Programming, Vol. 25,*

No. 1, 1995.

3. *RTCA/DO-178B (1992): Software Considerations in Airborne Systems and Equipment Certification. Requirements and Technical Concepts for Aviation.*
4. *Michael R. Donat, "Automating formal specification-based testing", In Michel Bidoit and Max Dauchet, editors, TAPSOFT `97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, volume 1214 of Lecture Notes in Computer Science, SpringerVerlag, April 1997.*
5. *John Rushby, Automated Test Generation And Verified Software, Computer Science Laboratory, SRI International.*
6. *Andy Galloway, Frantz Iwu, John McDermid and Ian Toyn, "On the Formal Development of Safety-Critical Software", Department of Computer Science, University of York, Heslington, York, UK*
7. *Michael R. Donat, "Capturing the Logical Structure of Requirements for the Automatic Generation of Test Specifications"*
8. *David LorgeParnas, "Requirements Documentation: A Systematic Approach", Department of Computer Science and Information Systems, University of Limerick*
9. *Tony Hoare, "The Verifying Compiler: A Grand Challenge for Computing Research", Microsoft Research Ltd.*
10. *CAR Hoare, The Verifying Compiler: a Grand Challenge for Computer Research, JACM (50) 1, pp 63–69 (2003)*
11. *Cliff Jones, Peter O'Hearn, Jim Woodcock, "Verified Software: A Grand Challenge"*
12. *NASA. 1997. Formal methods, specification and verification guidebook for the verification of software and computer systems. vol II: A practitioner's companion. Tech. Rep. NASA-GB-00197, Washington, DC. May.*
13. *NASA. 1998. Formal methods, specification and verification guidebook for the verification of software and computer systems. vol I: Planning and technology insertion. Tech. Rep. NASA/TP98-208193, Washington, DC. Dec.*
14. *Michael Jackson, "Formal Methods & Traditional Engineering"*
15. *JING SUN, JIN SONG DONG, JING LIU and HAI WANG, "A Formal Object Approach to the Design of ZML", Department of Computer Science, School of Computing, National University of Singapore*
16. *ISO, Information Technology, Z Formal Specification Notation, Syntax, Type System and Semantics, ISO/IEC 13568:2002*
17. *J. M. Spivey, "The Z Notation: A Reference Manual", Second Edition, Programming Research Group, University of Oxford.*
18. *K. Thangavelu, "Control System Software Execution During Fault Detection," 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2022, pp. 1-5, doi: 10.1109/ICICCS53718.2022.9788193.*
19. *Thangavelu, Kamaraj, Framework for Data Management System to Assist Aircraft System Maintenance (June 24,*

2022). IJCRT | Volume 10, Issue 6 June 2022, Available at SSRN: https://ssrn.com/abstract=4145757

20. *Ziqiang Yin1, a, Xinqiang Ma, b, Xiao Zhen, c, Wenlong Li , b and Wei Cheng, "Welding Seam Detection and Tracking Based on Laser Vision for Robotic Arc Welding", (2020) .*
21. *Y K. H. Li, J. S. Chen, and Y. M. Zhang in their research "Double-Electrode GMAW Process and Control" Toyota Motor Manufacturing North America, Inc.*
22. *Dingjian Ye, Xueming Hua, and Yixiong Wu Research Article mention "Arc Interference Behavior during Twin Wire Gas Metal ArcWelding Process" (2013).*
23. *Robert Bogue/Industrial Robot The international journal of industrial and service robotics/Emerald Publishing Limited 06/10/2019*