Power and Limitations of Formal Methods for Software Fabrication: Thirty Years Later

Edgar Serna M. and Alexei Serna A.
Facultad de Ciencias Básicas e Ingeniería, Corporación Universitaria Remington. Medellín, Antioquia, Columbia
E-mail: edgar.serna@uniremington.edu.co, alexei.serna@uninremington.edu.co

Keywords: formalization, automation, formal languages, software lifecycle, software quality

Received: February 16, 2017

In 1987, Michael Jackson presented his work "Power and Limitations of Formal methods for software fabrication" at the AIT Conference, which analyzed the advantages and limitations of formal methods up to that time. His conclusion was that formal methods had undoubted capabilities and advantages, but they also had serious limitations that prevented their widespread acceptance and adoption. The aim of this paper is to present the current context of formal methods compared with what Jackson described three decades ago. A tour of the strengths and limitations of formal methods is taken through a review of literature in the timeline of the past thirty years. The conclusion is that little progress has been made on this issue in relation to the situation presented by Jackson, and formal methods still need more work from academia, industry and the community.

Povzetek: Prispevek analizira napredek formalnih metod s primerjavo z Jacksonovo metodo izpred trideset let.

1 Introduction

The idea of making mathematics an area of increased use and applicability in different disciplines and contexts can be traced to ancient Greece, where Pythagoras, Plato, Aristotle and Euclid tried to make its study and use accessible to a wide audience [1]. Beyond incipient astronomy, the development of physics, public works and the little there was of mechanics, however, its context continued to be limited to accounting and commercial calculations [2].

For a long time, initiatives were developed with similar objectives, and although some have been relatively successful, especially with the emergence of the engineering disciplines and scientific specializations, the situation appears to remain in other areas [3]. Despite these achievements, the general idea is that mathematics is a field of knowledge that is extremely complicated and difficult to learn and apply to social realities. This attitude has created many myths that have taken hold in the formative process, where students manifest a fear of taking mathematics courses whose content is higher than that in other courses [4, 5]. Beyond the fact that these myths may or may not be true, the reality is that there is still no generalized context in which mathematics is appreciated for what it is and not what it seems. For example, one can mention that without the contributions of mathematics, major scientific and engineering developments would not have materialized in areas such as astronomy, physics, chemistry, natural sciences and, more recently, computer science [6]. In the latter, it was adopted as formal methods with the idea of mathematizing processes to develop software and design hardware [7].

Although many moments of the appearance of formal methods can be found in the history of these sciences and many authors have submitted contributions in this regard, it was not until the 1960s that the concept was taken seriously, after the enactment of the so-called software crisis [8]. The community then directed its gaze to mathematics as a lifeline that had helped other disciplines, with the aim of integrating it into software development to solve this crisis and ones that might appear later.

Since that time, various researchers, scientists, authors and organizations have been given the task of mathematizing software development and automating their tests in search of better-quality products. At this time there has been progress, but at the same time, many problems have been found due to the limitations diagnosed in formal methods [9]. In 1987, Michael Jackson [10], a British computer scientist and professor, made a presentation on what he considered the advantages and limitations of formal methods at that time. Three decades after his presentation, it is time to review whether mathematics in computer sciences has overcome those weaknesses and built upon its strengths, if it continues on the same path, or if it needs more time to achieve what was proposed as a lifesaver for software problems.

Building on the work of Jackson, the aim of this article is to take a tour through three decades of publications on the advantages and limitations of formal methods and determine how far we have advanced in their potentiation and/or improvement. This work presents what authors have proposed, innovated and applied regarding the formalization of software development, and the results and conclusions of Jackson are contrasted with the current reality. In addition, current and future challenges for industry, academia and the community regarding the acceptance and widespread use of formal methods are described.

2 Method

To develop this review, it was applied the methodology proposed by Serna [47] to perform reviews of the literature. The search was performed in the following databases: ScienceDirect, ACM Digital Library, Scopus, and Web of Science. It was searched the term Formal Method(s) combined with the terms definition, description, power, limitations, best practices, effective development, and development, first in the title or abstract. By applying this method were selected 78 works including articles, books, works in events and websites. To this population, the sample was applied the inclusion/exclusion criteria (thematic pertinence, author's relevance, focus, quality of results, practical application, and others) in order to determine if the content contributes to the achievement of the goals set in this review. After this procedure, we get 48 works, and after performing a quick reading and applying the concepts of quality in order to determine the value of each of them for this research the final sample was constituted by 40 works.

3 Jackson's conclusions

The work of Jackson [10] had the goal of presenting an analysis of the state of formal methods for the decade of the 1980s. In his presentation, he argued that it would be tedious and boring to describe the advantages and limitations alone, and for this reason, he dedicated almost all of the content to analyzing the fact that the problem was not so much with the formal methods but rather with the body of knowledge itself or the practice of developing software at that time. He asserted that these limitations could not be overcome solely by improving formal methods because they had been imposed by the inherent informality of that practice. To this end, it was one thing to describe the real world with mathematics and a very different thing to do so with natural language because the ambiguity of the latter creates complications for translation.

He also argued that formal methods offered a range of formalization but did not indicate which option to select nor how to apply it in the development of software, a task that corresponded to the developer based on experience and skills. For him, at that time, it was thought that software development was like a manufacturing process, in which descriptions were written in some language and assumed to be analogous to the parts of a mechanical product, where language was the raw material to manufacture them. It was also assumed at that time that software development was primarily a task of composition, not of decomposition, which duplicated the thinking in the forties of engineers in the construction of rockets, who felt that the process consisted of five descriptions: a guidance system, a propulsion system, fuel, a structure and aerodynamic principles, when, to satisfy them all, only the structure, fuel and streamlining were needed. According to Jackson, that process is what defines composition, but it demands creativity and invention that cannot always be automated.

He concluded that formal methods have undeniable advantages and potential, but for the practice of software development at the time, they also had serious limitations that hampered their widespread acceptance. Although most of his work was devoted to demonstrating that most of the blame belonged to the practice of software development, he listed some advantages and limitations, which are presented in Table 1.

Power
Their descriptions are accurate and non-ambiguous Their descriptions can be manipulated by symbols Mathematics provides a high degree of reliability The math is based on a large body of knowledge
Limitations
They restrict the developer to a single language They are focused on transformations between descriptions They tend not to be methods Not all software projects can be formalized Formalisms tend to be isolated from each other Research focuses on individual formalisms A broad integration of formalisms is required

Table 1: Power and limitations of formal methods [10].

Three decades have passed since these claims, and there remains in the environment the feeling that formal methods cannot become an alternative for developing reliable, secure and quality software. In the next section, the development of formal methods over the 30 years after the work of Michael Jackson is described.

4 The last thirty years of formal methods

In Seven myths of formal methods, Anthony Hall [11] presents his analysis of seven myths that existed at that time: 1) they ensure that the software is perfect, 2) they prove that the program is correct, 3) they are only useful in critical systems, 4) they involve complex mathematics, 5) they increase the cost of development, 6) they are incomprehensible to customers, and 7) nobody uses them in real projects. The author claims that many of the things said for or against formal methods were generated from the experiences of developers when applying them but that, although there might be some uncertainty, the reality was that they had more advantages than disadvantages. In his own experience, said Hall, these myths had to be reformulated and established as a type of process because, by then, the transfer from academia to industry was working consistently. To Gaudel [12], the advantages and problems of formal methods were limited to specification and design, as shown in Table 2.

According to Young [13], positive or negative opinions on formal methods had generated controversy for years, while the formal methods community fell short in explaining what they are and what their advantages are due to using descriptions and language that only community members understood and recognized. However, likewise, he asserted that there was a lack of will in the software engineering community to recognize the true value of formal methods. For him, if the goal was to improve the quality of software, it was necessary for both communities to work together to achieve it. For their part, Barroca and McDermid [14] felt that formal methods could be used in two different ways: to produce specifications for conventional systems development and, from that point, to generate formal specifications to verify the accuracy of the program. For these authors and at that time, the benefits of formal methods were as follows: 1) they ensure a consistent interpretation of the specification, 2) they allow verification of the application, and 3) they remove the ambiguity of language. They also listed weaknesses: 1) their development status is low, 2) the specification cannot be validated, 3) they only have mathematical interpretations, and 4) non-functional requirements cannot be adequately articulated in the context.

	Power	**Limitations**
Specification	They make it possible to analyze and encourage it It is structured and reusable It is testable	Correctness cannot be formalized To express the properties, it is necessary to formulate certain aspects of the application domain They only allow external verification
Design	It is the best way to prevent human errors It is a good approximation to zero failures Correctness tests are improved	They are difficult to implement It is still necessary to check the design with a tool The mathematical rigor does not completely eliminate errors The tests do not provide security

Table 2: Power and limitations of formal methods [12].

For Robert Vienneau [15], changes in computing in the 1970s and 1980s generated revolutionary ideas that materialized in formal methods, but there was not yet a unified philosophy about them. Although formal methods were promulgated as a technology applicable to the entire software life cycle, the author wondered why they were not more widely known. He asserted that part of the problem was educational and that many of its limitations would never be overcome, but he was convinced that some restrictions would be addressed through research and practice. Table 3 shows the limitations and advantages that this author described.

Power
They can be used to verify a system They complement the natural language descriptions and give them accuracy They can show that an implementation satisfies a specification More precise specifications are achieved Internal communication is improved They provide the ability to verify designs before running them during the test They offer higher quality and productivity
Limitations
They cannot be used to validate a system They can never replace the knowledge that the engineer has of the system They can never fully replace tests Their applicability is doubtful in systems with many lines of code

Table 3: Power and limitations of formal methods [15].

Liu and Adams [17] concluded that developers utilized formal methods with the hope of refining processes and improving specifications but often did not achieve their goals due to the limitations of this technology: the refinement rules are not sufficient to guarantee that a refined specification satisfies the requirements, and in addition, these rules cannot be reutilized and are difficult to apply in practice. For this reason, they recommended modifying the existing refinement rules, if the objective is to make formal methods widespread. According to Craigen et al. [18], at that time, formal methods were a developing technology, and therefore, they exhibited limitations, like any other such technology. Moreover, it was necessary to determine two key aspects: 1) what were the boundaries between the real world and the world of mathematics and 2) what were the internal limitations of the mathematics. They felt it was difficult to address these issues because, at that time, research placed the real world in doubt; therefore, mathematizing the needs of the client became an informal process. This limitation hindered the widespread growth and recognition of formal methods among software professionals.

Bowen and Hinchey [19] asserted that, for some reason, in the 1990s, formal methods had become one of the most controversial techniques in software engineering. They took as a foundation the work of Hall [12], and they added perspectives that they considered to be new myths: they delay the development process, tools do not support them, they are not really methods, they only apply to software, they are not necessary, they do not have support, and the formal-methods community always uses formal methods. For them, the problem was that more real relationships between academia and industry were required, it was necessary to spread experiences (positive and negative) by using them more widely, more research was needed, and it was necessary to demystify mathematics. Rushby [20] said that, in that decade, formal methods had progressed from an academic curiosity to an industrial reality, and he presented an analysis of their past, present and future. He concluded that to achieve widespread adoption, it was necessary to improve the tools and the scale of their applications, that theoretical research should provide better characterization, and that the software industry must have an open mind regarding this technology because the hardware industry was already enjoying its benefits.

Steve Easterbrook [21] described three case studies in which they applied formal methods to model requirements. They argued that, in contrast to other projects in which Requirements Engineering was used very early on to validate needs, in their experience, they were able to improve the specification. They concluded that the benefits of formal modeling are that it reduces process costs, it enables more effective verification and validation, and maintenance is structured better. Meanwhile, for Kneuper [22], formal methods could help improve the reliability of software development but did not solve all problems. The author described the limitations that make universal solution by formal methods impossible: complete formalization is not possible, there is no guarantee that the informal user requirements are correct and complete, it is difficult (almost impossible) to ensure that the program is correct, they do not determine correct tests, abstraction does not accurately reflect the application, they are applied on a small scale, the technical development is insufficient, and developers do not have the proper mathematical training.

According to John Knight and his team [23], by that time, formal methods had proven benefits, but there were several reasons they lacked broader acceptance: they extend the development cycle, they require complicated mathematics, and the existing tools are inadequate and incompatible with other software packages. However, after applying formal methods to critical application specifications, they concluded that, although several of those reasons could be valid, the main issue was to try to build a comprehensive evaluation framework for the specification. Given that until then, it had not been achieved, it became a stumbling block that the industry could not solve, but given the orientation of the subsequent research, it could be solved in future work. For Jeannette Wing [24], formal methods had limitations in ensuring the security of systems, but they delineated the boundaries of the systems and characterized their behavior more accurately, defining their desired properties with precision and providing a specification in terms of time. She explained that their limitations were because the system operates in an environment, and therefore, formal methods cannot provide total security. She also said that the future of formal methods was promising because, in that decade, many research initiatives were conducted.

Jones et al. [25] conducted research on the contributions of formal methods to requirements engineering and found that some challenges still remained to be overcome: how to couple informality to the formality of requirements, better manage changes, allow traceability, improve accuracy in the validation of the specification, offer better alternatives for non-functional, reconcile some inconsistencies in notations, allow multiple notations and create a body of knowledge that includes all parts involved in this phase. In the same vein, Heylighen [26] stated that the validation of knowledge requires formal expressions of the same and that mathematical determinism requires greater co-pagination with operational determinations. He believed that any formalization process has advantages: it removes ambiguity, it defines the extension of terms, it is independent of time, mathematical language is universal, and it is reusable and testable. However, it also has limitations: it is generally isolated from the context, it has intrinsic limitations, and it assumes normal conditions as implicit contexts, whereas causal factors determine context dependence. He predicted that it was necessary to overcome the limitations and potentiate the advantages to popularize the formalization of software.

Power
They provide units of measure They facilitate the detection of errors They ensure proper operation They reduce errors They help improve abstraction They perform rigorous analysis They are reliable They allow effective test cases
Limitations
They require informality to guarantee the specification It is not easy to see that the implementation satisfies the specification It is not possible to guarantee that the tests are correct The language features are complex Technical environments do not always recognize a formal specification

Table 4**:** Power and limitations of formal methods [37].

Wordsworth [27] summarized the benefits of formal methods as follows: 1) successful cases have been sufficiently reported, and 2) although the basis is mathematical, it is not always necessary. His caveats, however were that formal methods demand some degree of mathematical sophistication, they are not taken seriously in programs of study, users are satisfied with traditional methods, the requirements should be specified more precisely, and developers prefer to code without complete specification. According to [28], formal methods had not yet achieved greater penetration; a wide gap persisted between research in academia and industry application. They maintained that the fact that industry still did not believe in formal methods was due to the loss of scalability, limited access to specialists and the immaturity of tools and techniques.

Peter Amey [29] presented what he called the reality of formal methods and described a series of cases of successful software development. He inquired why, despite its utility, the approach still was not widely used in industry, and he concluded that it was because, typically, they were trying to use development at inopportune times. That is, the error was not how but when. He suggested it would be advisable to start with specification and then reach verification and validation. Edmonds and Bryson [30] argued that the idea of formal methods offered two advantages: 1) the specification is unambiguous, and 2) it can be self-handled syntactically. However, they felt that formal language presented difficulties when trying to translate to or from other languages, which generated two problems: 1) natural language translation is slow, and 2) coding is delayed. Martin Gogolla [31] summarized the benefits and problems of formal methods through a literature review and classified his findings based on indicators: domain of application, persons, properties, tools, understanding, development and general criticism. He concluded that the success or failure of formal methods was not determined by their mathematical properties but by the low usability of existing tools.

For Glass [32], formal methods had existed for a long time but still did not achieve the impact they should have, even though the specification is considerably more understandable, the confidence of the analyst increases because they identify the key problem to solve, errors in the final product are reduced, and maintenance costs are reduced due to the knowledge acquired. Hall [33] attempted to demonstrate the benefits of formal methods and asserted that achieving them was not automatic because there was no better way nor better method to do so. He thought that they were only part of the solution to the problems of software development and that their success depended largely on a clear integration: that intelligent use is required, that researchers should dedicate more time to them, that practical issues of integration and access are as important as the theoretical issues, and that developers must let go of the fear of formalisms. Sommerville [34] wrote that since the 1980s, many engineers and researchers had proposed the use of formal methods as the best way to improve the quality of software products, but that dream still had not come true. He concluded that there were four reasons: 1) the emergence of new methods and management proposals that have helped improve the quality and success of Software Engineering; 2) a new market where quality seems to be of secondary importance; 3) the limited reach of formal methods, which still do not adequately exceed specifications; and 4) limited scalability because large projects are still not satisfied. Still, for him, formalization was an excellent way to discover errors in specification.

David Parnas [35] argued that in the last 40 years, three alarming gaps had appeared in the software field: 1) between research and practice, 2) between software development and traditional engineering disciplines, and 3) between computer sciences and classical mathematics. He argued that advocates of formal methods proposed them as the solution to any of those gaps, even though, up until that time, they could not be verified. He concluded that formal methods had been left with only that perspective, and it was time to rethink them. To achieve this rethinking, he proposed the following: 1) software has problems, but some formal methods with problems will not solve them, 2) more research is needed as is fewer defensive efforts, 3) movement should be slow, not all at once, 4) abstractions should be simple, but true, and 5) our role in the model must be as engineers, not as philosophers and logicians. For the IET [36], formal methods offered the following advantages: they allow a consistent and reasonably complete specification, they reduce the likelihood of error and the cost of detection, and they permit identifying ambiguity in the specification and verification of security requirements. Nonetheless, they were not widely used because the industry did not give them real opportunities and because academia did not view them seriously.

In the article by Batra et al. [37], it became clear that by then, the demand for incorporating formalization in Information Systems had increased because the specification represented actual requirements, and the formal methods could ensure that the implementation met the specifications and demands of security, reliability and quality, although they also had weaknesses. Table 4 describes the advantages and limitations of formal methods for these authors.

In the results of a survey conducted by Fitzgerald [38] on the impact of formal methods on the cost, time and quality of the software, the opinions were divided: 25% reported a decrease and 20% an increase in time; in terms of cost, 33% said there was a reduction and 8% said it increased; and in quality, 88% asserted that it improved and 8% were not sure. In [9] identified eight obstacles to the research, teaching and practice of formal methods: 1) there is insufficient research and teaching, 2) support tools are not sufficient for use on a large scale, 3) students are not taught Computer Science or Software Engineering in mathematical terms, 4) no foundations are strengthened and no attempt is made to present new functions, 5) there are not enough graduates with mathematical knowledge to serve the industry, 6) professors of computer science and Software Engineering do not receive the same training they did 30 years ago, 7) formal methods lack tools for managing versions and configuration control, and 8) professionals who are trained in formal methods do not find support among their industry fellows, so they tend to abandon the practice.

Ishikawa et al. [39] stated that formal methods were increasingly attracting more attention as a solution to the high demand for efficient and reliable software, but a gap had developed between knowledge and teaching with which software engineers were educated and what was required to implement these methods. Meanwhile, for Mayo et al. [40], the limitations of formal methods lay in semantics and traceability, as software and hardware are formally tested during the development process only for explicit statements made ahead of time and as the requirements are validated only in the semantics in which they were tested. According to Gross et al. [41], the exhaustive testing of software systems is intractable and expensive, but if formal methods are incorporated throughout the design process, errors can be identified as they are introduced and the total cost of development dramatically reduced.

5 Analysis of results

After presenting a review of the timeline of formal methods in the past 30 years, looking for the advantages and limitations published by various authors, it is difficult to determine whether the advantages of formal methods have been improved or new ones found. Regarding the limitations, it cannot be stated clearly whether they have been overcome or if instead they have become more acute or others have appeared. The conclusions and presentations in these decades are divided between positivism and negativism toward formal methods, even predicting its possible disappearance from the software development stage. In any case, after analyzing these conclusions, the map of reality of formal methods in these three decades can be seen from three dimensions: from academia, from the community and from the industry. Next, we consider each from the perspective of the results.

In the academic dimension, many works report that formal methods do not achieve the expected penetration because the curriculum in Computer Science and Software Engineering still does not pay adequate attention to them. Thus, these professionals are not educated in applied mathematics, and the few who are have not found peers in industry interested in sharing their knowledge. In this sense, academia should provide an opportunity for formal methods and include them in its content, and in addition, professors need more training to avoid improvising in the classroom. While software problems will not be solved in this way from one moment to another, if progress has already been made, it is better to exploit formal methods to see if they can improve software quality [42].

Furthermore, education systems should take responsibility for the fact that students have mythicized mathematics because they are structured to educate everyone equally and in all areas. With this approach, the skills that each individual may have for one or another discipline are wasted because they do not receive a vocational orientation that tells them how to orient their educational needs. The reality is that to understand mathematics, one must first develop logical and abstract reasoning, but education systems have not contemplated this foundation. Hence, the student prefers more theoretical or less logical areas because they require less effort. This reality is combined with the fact that software development is highly abstract because it is a non-tangible engineering product, and only the outputs and not the processes can be observed. This characteristic differentiates it from other products, such as civil engineering, in which the manufacturing process is constantly evident. The result is that professionals are not adequately trained in mathematics, and thus, formal methods are not practiced nor experienced to exploit their advantages and overcome their limitations.

In the community dimension, the opinion is reiterated in the literature review that formal methods have demonstrated their power in the formalization of specification. For many authors, Requirements Engineering is where formal methods have had the greatest acceptability and where the most success stories are found. The effect has been that the community has devoted less effort to further strengthening procedures and tools to elicit and specify requirements, dedicating itself instead to possibilities in other phases of the life cycle. Some authors criticize this trend in that the community believes that formal methods have already exceeded their goal and that thus another goal should be developed, when the reality is that there are still many problems in the formalization of specification. One recommendation is that what has been achieved so far with requirements should first be strengthened and standardized, and other possibilities can then be considered.

Another issue with the formal methods community is that it is perceived as closed to the participation of other stakeholders because language has limited its communication to the strictly mathematical [43] and because transdisciplinary work is not considered as an alternative. With the development of different disciplines, many researchers interested in contributing from their specialty to the development of other specialties have appeared, as in the case of Neurocomputation for understanding the brain and how people learn. If the formal methods community were more open to contributions from areas such as philosophy, psychology or didactics, it could achieve better results than it has so far.

On the other side is the software industry, a crucial factor in the current map of the reality of formal methods. For years, software was developed in laboratories and with military support because its potential was considered only from that perspective. Over time, society realized that software could be expanded as a solution to other needs, commercialization began, and the software industry appeared with the aim of development for sale. However, software was nonetheless adopted and adapted with the methodology that military scientists had built in their laboratories and applied to develop the products offered on the market. This approach to software development triggered what NATO deemed a crisis in the late 1960s. It is understandable that industry has intended to mainly produce to sell and make a profit, but software is a product that is not manufactured but rather is created (developed); therefore, different procedures are needed from the ones used, for example, to manufacture an aircraft or turbine.

With respect to formal methods, the industry still does not assimilate them at their full potential because it believes that they delay processes and reduce usefulness. The issue is that, if not in industry, where can the proposals of the community and of academia be verified and validated? The three dimensions must work in unison so that a software-dependent society can enjoy better-quality software products. This need does not mean that formal methods are the immediate and magical solution to the software crisis, but as an alternative, it is worth the effort to give them an opportunity while there is no other alternative.

6 Conclusions

The increasing complexity of systems in this century is a challenge for research in Computer Science. The hardware and software that make up these systems have gone in a few years from a few components and lines of code to hundreds of thousands. One need only compare the reality that Jackson described in his work, three decades ago, with the one in which we currently live, in the midst of a software-dependent society with high demands for quality, reliability and product security. However, the software component costs half or more of the total value of the development of a system, and its applications impact economies around the world. For all these reasons, it is necessary to innovate with regard to development processes because, although many think otherwise, we still have not overcome the so-called software crisis of the 60s.

Two key issues can be identified from this analysis of the reality of formal methods in the past three decades. 1) The processes for developing software are still performed as they were more than 50 years ago. Very little innovation has occurred in this sense, and interest seems geared more to proposing and selling new languages and methodologies than to positioning and strengthening the ones that exist and have been proven to work. New approaches are not always the solution, and often what is achieved is to increase the range of options but hinder the work of developers. 2) Formal methods are mathematical and therefore are not yet widespread. In this sense, we must understand that mathematics is based on the understanding and application of logic and pure abstractions, and although computers exist in physical reality, software is basically responsible for representing and manipulating non-physical data. That is, mathematics represents the physical reality, while software models and simulates it.

Formal methods were developed over decades and have introduced principles, paradigms and influential conceptual innovations into computer science for the development software. This is reflected in the fact that a quarter of the Turing Awards between 1966 and 2013 recognize work with a significant component in formal methods. Nonetheless, they still seem to be at a crossroads: as an advantage, they seem well developed and are supported by a large number of applications, users and important critical developments; however, as a limitation, they have ceased to be a major component in computer science and engineering training, few professors are working on them, course offerings at the undergraduate and graduate levels are scarce, they are difficult to apply in important projects, and only a small number of graduates welcome them as a source of work. Moreover, education systems do not adequately develop the logical interpretation and abstracting ability of students [44, 45], which are necessary for logical reasoning. This area of study even seems to have decreased in recent years, which has made new generations increasingly prejudiced regarding mathematics [46].

Thirty years after the work of Michael Jackson, the outlook for formal methods seems not to have changed much. The advantages and limitations that this author described remain, and others seem to have become more acute because variables entered the scene that at the time were unknown: the abandonment of mathematics as the center of the universe of educational systems, the social demand for new and innovative products in a very short time frame, the increasing complexity of problems and the emergence of tools that attempt to displace developers, among others. The facts that the community of formal methods remains isolated from the reality of Computer Science and that the industry does not provide the necessary space to popularize its application are also of little help.

We can thus conclude that, as a solution to the problems of software development, formal methods still have a long way to go. Work over the past thirty years has been slow and achievements few. It has not been possible to form a suitable environment to establish formal methods as an area of academic and industrial interest, and professors have lacked the training and experience to include them in the curriculum. In addition, students still perceive mathematics as an obstacle to be overcome to graduate, rather than as an important part of the learning process. If the formal methods community is integrated and works with other knowledge disciplines, if the industry works a little more and if academia grounds its theories in an attempt to overcome these limitations, formal methods could constitute an alternative way for software to achieve the security and quality expected by society.

7 References

[1] Cohen, B. (1995). A brief history of ‘Formal Methods’. *Formal aspects of computing,* 1(3), 1-10.

[2] Neugebauer, O. (1969). *The Exact Sciences in Antiquity*. USA: Dover Publications.

[3] Serna, M.E. (Ed.) (2015). *Avances en ingeniería*. Medellín: Editorial Instituto Antioqueño de Investigación.

[4] Dani, S. (1993). 'Vedic Mathematics': Myth and Reality. *Economic and Political Weekly*, 28(31), 1577-1580.

[5] Dowling, P. (1998). The Sociology of Mathematics Education: Mathematical Myths/Pedagogic Texts. London: Falmer Press.

[6] Holloway, C. (1997). Why Engineers Should Consider Formal Methods. *16th Digital Avionics Systems Conference*. Irvine, USA, 16-22.

[7] Butler, R. (2001). What is Formal Methods? NASA. Online [Aug 2016].

[8] Naur, P. & Randell, B. (1968). Software Engineering. *Scientific Affairs Division NATO*. Germany, Garmisch.

[9] Bjørner, D. & Havelund, K. (2014). 40 years of formal methods- Some obstacles and some possibilities? *Lecture Notes in Computer Science*, 8442, 42-61.

[10] Jackson, M. (1987). Power and limitations of formal methods for software fabrication. *Journal of Information Technology,* 2(2), 1-6.

[11] Hall, A. (1991). Seven Myths of formal methods. *IEEE Software*, 7(5), 11-19.

[12] Gaudel, M. (1991). Advantages and limits of formal approaches for ultra-high dependability. *Proceedings of the Sixth International Workshop on Software Specification and Design*. Como, Italy, 237-241.

[13] Young, W. (1991). Formal Methods versus Software Engineering - Is there a conflict? *Fourth Testing,*

Analysis, and Verification Symposium. Victoria, Canada, 188-899.

[14] Barroca, L. & McDermid, J. (1992). Formal Methods: Use and relevance for the development of safety critical systems. *The Computer Journal*, 35(6), 579-599.

[15] Vienneau, R. (1993). *A review of Formal Methods*. Technical Report, Kaman Science Corporation.

[16] Gerhart, S., Craigen, D. & Ralston, T. (1994). Experience with Formal Methods in Critical Systems. *IEEE* Software, 11(1), 21-28.

[17] Liu, S. & Adams, R. (1995). Limitations of formal methods and an approach to improvement. *Proceedings Asia Pacific Software Engineering Conference*. Brisbane, Australia, 498-507.

[18] Craigen, D., Gerhart, S. & Ralston, T. (1995). Industrial applications of formal methods to model, design and analyze computer systems - An international survey. New Jersey: Noyes Data.

[19] Bowen, J. & Hinchey, M. (1995). Seven more myths of formal methods - Dispelling industrial prejudices. *Lecture Notes in Computer Science,* 873, 105-117.

[20] Rushby, J. (1996). Mechanized Formal Methods: Progress and prospects. *Lecture Notes in Computer Science*, 1180, 43-51.

[21] Easterbrook, S. et al. (1996). *Experiences using formal methods for requirements modeling*. Technical Report NASA-CR-203085.

[22] Kneuper, R. (1997). Limits of Formal Methods. *Formal Aspects of Computing*, 3(1), 1-16.

[23] Knight, J., Dejong, C., Gibble, M. & Nakano, L. (1997). Why are formal methods not used more widely? *Fourth NASA Langley Formal Methods Workshop*. Virginia, USA, 1-12.

[24] Wing, J. (1998). A symbiotic relationship between formal methods and security. *Proceedings Conf. on Computer Security, Dep. and Assu: From Needs to Solutions*. Williamsburg, USA, 26-38

[25] Jones, S., Till, D. & Wrightson, A. (1998). Formal Methods and Requirements Engineering - Challenges and Synergies. *Journal of Systems and Software*, 40(3), 263-273.

[26] Heylighen, F. (1999). Advantages and limitations of formal expression. *Foundations of Science*, 4(1), 25-56.

[27] Wordsworth, J. (1999). Getting the best from formal methods. *Information and Software Technology*, 41, 1027-1032.

[28] Broadfoot, G. & Broadfoot, P. (2003). Academia and industry meet - Some experiences of formal methods in practice. *Tenth Asia-Pacific Software Engine. Conference*. Chiang Mai, China, 49-58.

[29] Amey, P. (2004). Dear Sir, yours faithfully - An everyday story of formality. In Redmill, F. & Anderson, T. (Eds.), *Practical Elements of Safety*. UK: Springer, 3-15.

[30] Edmonds, B. and Bryson, J. (2004). The insufficiency of Formal Design Methods. *Proceedings Third International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, USA, 938-945.

[31] Gogolla, M. (2004). Benefits and problems of Formal Methods. *LNCS*, 3063, 1-15.

[32] Glass, R. (2004). The mystery of Formal Methods Disuse. *Communications of the ACM*, 47(8), 15-17.

[33] Hall, A. (2005). Realising the benefits of Formal Methods. *LNCS*, 3785, 1-4.

[34] Sommerville, I. (2009). *Software Engineering*. New York: Pearson.

[35] Parnas, D. (2010). Really rethinking 'formal methods'. *Computer*, 34(1), 28-34.

[36] IET. (2011). *Formal Methods*. The IET.

[37] Batra, M., Malik, A. & DAVE, M. (2013). Formal methods - Benefits, challenges and future direction. *Journal of Global Research in Comp. Scien.*, 45, 21-25.

[38] Fitzgerald, J. (2013). Industrial deployment of formal methods: Trends and challenges. In Romanovsky, A. and Thomas, T. (Eds.), *Industrial Deployment of System Engineering Method*s. Berlin: Springer, 123-143.

[39] Ishikawa, F., Yoshioka, N. & Tanabe, Y. (2015). Keys and roles of formal methods education for industry: 10 Year experience with Top SE Program. *Proceedings First Workshop on Formal Methods in SEE & Training*. Oslo, Norway, 35-42.

[40] Mayo, J., Armstrong, R. & Hulette, G. (2015). Digital System Robustness via Design Constraints: The Lesson of Formal Methods. *In 9th IEEE International Systems Conference*. Vancouver, Canada, 109-114.

[41] Gross, K., Fifarek, A. & Hoffman, J. (2016). Incremental Formal Methods Based Design Approach Demonstrated on a Coupled Tanks Control System. *Proceedings 17th International Symposium on High Assurance Systems Engineering*. Orlando, USA, 181-188.

[42] Alvear, A. & Quintero, G. (2015). Integrating software development techniques, usability, and agile methodologies. *Actas de Ingeniería* 1, 94-103.

[43] Polansky, J. & Sinclair, M. (2014). The importance of training in formal methods in Software Engineering. *Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de Software (RACCIS)*, 4(2), 52-56.

[44] Serna, M.E. (2013). *Prueba funcional del software - Un proceso de Verificación constante*. Medellín: Editorial Instituto Antioqueño de Investigación.

[45] Serna, M.E. & Serna, A.A. (2013). Is it in crisis engineering in the world? A literature review. *Revista Facultad de Ingeniería*, 66, 197-206.

[46] Tucker, A., Kelemen, C. & Bruce, K. (2001). Our curriculum has become Math-Phobic! *Proceedings 32th Technical Symposium on Computer Science Education*. Charlotte, USA, 243-247.

[47] Serna, M.E. (2016). Methodology for perform reliable literature reviews. *Revista Investigación Económica*, in press.