Software Validation

When 'Pure Mathematical Objectivity' is no Longer Enough

Isabel Cafezeiro
Instituto de Computação
Universidade Federal Fluminense
Niterói/
Programa de História das Ciências e das Técnicas e
Epistemologia (HCTE-UFRJ)
Brasil
e-mail: isabel@dcc.ic.uff.br

Ivan da Costa Marques
Programa de História das Ciências e das Técnicas e
Epistemologia (HCTE-UFRJ)
Universidade Federal do Rio de Janeiro
Rio de Janeiro,
Brasil
e-mail: imarques@ufrj.br

Abstract— **By focusing on systems that can be trusted to operate as required, software validation offers a rich field to study how far one can go with the support of mathematical certainty, that is, to identify when evidence (a non formal entity) must come into play to dismiss the possibility of critical errors. First, this article highlights that the view of mathematics as a source of accuracy supported by a purified and rational chaining of reasoning persists until the present days. Resorting to historical controversies of the 1970's regarding software validation, it is possible to indicate local (social) elements that necessarily participate in what is usually considered 'technical' or 'objective', showing therefore that there is no way to establish rigid or fixed boundaries delimiting what is considered 'exact'. Regarding software correctness, the sociotechnical approach adopted in this paper leads to a intertwined frame where social (collaborative) mechanisms act in ways that are inseparable from those mechanisms that are considered 'technical' or 'objective', which are, in this case, formal methods. This paper discusses software validation in the light of Sociology of Mathematics and Social Studies of Science and Technology.**

Keywords- formal specification; collaborative development; objectivity; sociology of mathematic.

I. INTRODUCTION

'The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness' [1]. Edsger Dijkstra, a spokesman of formal methods for software reliability in the seventies, argued in favor of a more rigid way to develop software, as a reaction to the just before denounced *software crisis* in the 1968 Conference on Software Engineering, Garmisch, reported in [2]. He defended that programs should be constructed 'hand-in-hand', module by module, with their formal proof. Dijkstra opposed his proposition to the traditional technique, 'to make a program and then to test it', which, in his view, was an effective way to detect the presence of errors but did not guarantee their absence. [1]

Dijkstra allied himself with a powerful partner: mathematics. Among mathematicians, and also in the common sense, there is a widespread culture of objectivity and accuracy of mathematics [3], which is strong enough to stifle dissenting voices, those sympathetic to non-formal mechanisms [4].

A. *The strengthening of mathematics to support the trust*

'Why are mathematical certainty and the evidence of demonstration common phrases to express the very highest degree of assurance attainable by reason?' [5]

It is not the purpose of this section to present an exhaustive historical account of the extensively expanded relations between mathematics and trust and certainty. Much more modestly, it brings in for discussion some historical moments over the last few centuries where those relations were under debate. The above quotation is a milestone: John Stwart Mill, in 'A System of Logic' (mid-nineteenth century) took a stand against the association of the highest degree of safety reachable by reason 'to mathematical certainty' and 'the evidence of demonstration'. This triggered intense objection by Gottlob Frege (Die Grundlagen der Arithmetik – 1884) [6], today considered one of the founders of modern logic, a spokesman for the strengthening of rationalist trend. The 'peculiar certainty' attributed to so-called 'deductive science' gained momentum in early twentieth century, through the Vienna Circle, where the logical positivists, declared their rejection to what they called theological and metaphysical speculation [7]. At the same time, amid the movements of mathematical foundations, in particular, David Hilbert's formalist program came to the fore. It conceived mathematics as a purely formal system, consisting of symbols devoid of meaning or interpretation: 'In a sense, mathematics has become a court of arbitration, a supreme tribunal to decide fundamental questions — on a concrete basis on which everyone can agree and where every statement can be controlled.' [8] The 1930s revealed surprises to these approaches, especially to the formalist program, with the publication of Gödel Theorems [9], which demonstrated the existence of statements that, although they could be written in a formal system of a certain kind, could not be proved in it. This exposed the inability of mathematics to decide any mathematically expressed matter maintaining its consistency. The publication of Gödel's theorems put into question the role of mathematics as a 'court of arbitration' as envisioned in the Hilbert's formalist program. Moreover, in

 ISBN: 978-1-61208-233-2

the 1940s some mathematicians have realized the need to consider factors then considered 'extra-mathematical' for understanding the configuration of mathematics itself. For example, the Dutch mathematician Struik proposed a 'Sociology of Mathematics' to be concerned 'with the influences of forms of social organization on the origin and growth of mathematical conceptions and methods' [10].

However, it is noteworthy that in the 1970s, Dijkstra proposed a program of mathematization of software by mobilizing arguments in bases that were very similar to those that David Hilbert had proposed in the formalist approach: the effort in the pursuit of mathematical truth and accuracy and consistency of mathematical methods. Even today, it is to be highlighted, the discourse of the search for security and reliability is widely supported by confidence in mathematics and formal systems. An illustrative example can be found in the general terms that conducted the formulation of 'The Grand Challenge Project' in 2005 attesting an enthusiastic view of formal methods: 'Programmers of the future will make no more mistakes than professionals in other disciplines. Most of their remaining mistakes will be detected immediately and automatically, just as type violations are detected today, even before the program is tested. An application program will typically be developed from an accurate specification of customer requirement; and the process of rational design and implementation of the code will be assisted by a range of appropriate formally based programming tools, starting with more capable compilers for procedural and functional programming languages.' [11]

The Sociology of Mathematics [12][10] allows us to question the 'objectivity' that is sought in mathematics and formal methods, highlighting that in its own conformation, mathematical entities are inseparably mixed with local or temporal elements, and are therefore historically situated. Such viewpoint takes social mechanisms for collaboration on a par with formal methods in the validation of software systems and reinforces the role of inductive reasoning, tests, empirical approaches as allies in the process of software validation.

B. Organization of the next sections

In Section II, we analyze the spreading of formal methods as a guarantee of software correctness since the 60's until today. In this process, we point out the inevitable presence of personal choices and subjectivities that the formal methods are unable to eliminate and the power of speech that relies on a so-called 'objectivity' of mathematics to enhance the trust in formal methods and the promise of a software free of errors.

In Section III, we present a case study relating mathematics and computers that reinforces the view that, when mathematics is requested to be applied in real-world situations, not only local issues are modified as a result of interaction with mathematics, but also mathematics changes as result of interactions with local issues. This view collapses with the general conception that math is unique and immutable as a 'language of Gods', a conception that persists not only in common sense, but also among mathematicians by cultural reasons. As expressed in David Hilbert's Radio Broadcast, in 1930: 'Already Galileo declared: "To understand nature, we must learn the language and the signs through which nature speaks to us." But this language is mathematics, and these signs are mathematical figures!'

In the case analyzed here, we report a new arithmetic - one that is remodeled by requirements of a computer hardware, what shows that in mathematics there is room for subjectivity. This example is a contribution to the understanding of how social elements come to be inseparable from the setting of mathematics, becoming part of it.

Then, in Section IV we return to the subject of software correctness. We consider arguments that emerged in the 1970s, in response to the mathematization of software. These reactions emphasized the importance of considering social mechanisms in the development of secure software and software verification. These social mechanisms gain a new dimension when we consider the new capabilities of interaction provided by the Internet, new techniques of software development considering collaboration and reuse and the speed of technology nowadays. Furthermore, we also consider a recent testimony in favor of the association between formal methods and empirical mechanisms. These are allies with whom formal verification can go far beyond.

We conclude this article indicating the Sociology of Mathematics and Social Studies of Science and Technology as emerging areas that consider the interweaving of mathematical knowledge and social and subjective issues. This kind of research supports mixed approaches in which recent mechanisms of collaboration can be taken in to build solutions to problems that were previously treated as purely technical.

II. Formal Methods for Software Correctness (from the seventies to today)

It was in terms of trust in mathematics in the late 1960s and early 1970s, when computer programs had become too long and were used in applications involving safety-critical situations, that the U.S. Department of Defense (DoD) initiated a series of debates that pointed to the mathematization of systems as a guarantee of correctness. At that time, the aim was to create a systematic methodology for building systems, to question the effectiveness of empirical tests and to bet on formal specifications as a means to enable secure programming in two ways: first, since the specification languages are more ‘abstract’ (more distant from the code that activates the hardware) than the programming languages, they can be closer to the problem domain, thus facilitating the correct understanding and ease of expression of the solution; and, second, since the specification languages are formal, they would be suitable to prove properties of programs, ensuring correctness. In 1985, the U.S. DoD published the Orange Book whose 'purpose [was] to provide technical hardware/firm-ware/software security criteria and associated technical evaluation methodologies', mandatory for use by all DoD Components in carrying out ADP (Automatic Data Processing) system. [13]

The establishment of these kinds of standards continued. In 1999 an arrangement of international organizations called Common Criteria (CC) created a basis for evaluating the security of information technology products, which then replaced the Orange Book. The CC defined seven levels of

assurance, (EAL), establishing a degree of trust directly proportional to an adherence to formal methods:

TABLE I. The Common Criteria Evaluation Assurance Levels: the more formal, the more reliable.

EAL1: Functionally Tested,
EAL2: Structurally Tested,
EAL3: Methodically Tested and Checked,
EAL4: Methodically Designed, Tested, and Reviewed,
EAL5: Semiformally Designed and Tested,
EAL6: Semiformally Verified Design and Tested,
EAL7: Formally Verified Design and Tested.

The role of formal methods in the view of Common Criteria can be understood from the report [14]: to earn certification the developer chooses and formalizes the properties he considers indispensable for safety, provides a formal specification of the parts of the software he considers critical and a proof that the chosen properties meet the specification. The last step is then to prove that the program is indeed a refinement (an implementation) of the given specification, and thus meets the properties proved at the formal level. These documents are then analyzed by the 'evaluation authority' – a team of specialists whose name reveals the sense of authority provided by mathematics.

As an example, we refer to [15], which describes 'how formal methods were used to produce evidence in a certification, based on the Common Criteria, of a security-critical software system'. This experience report makes clear that even being extremely formal the process always starts from choices, and these are inevitably subjective. As this report shows, the software developer chooses the pieces of code that are 'security-relevant software behavior'. He also decides which are the properties to be proved and where to locate the preconditions and postconditions in the code. However, what is considered difficult in the certification process are the formal steps, while the developer's choices are only briefly mentioned: 'Given 1) source code annotated with preconditions and postconditions and 2) a security property of interest, the overall problem is how to establish that the code satisfies the property. We developed a five-step process for establishing the property. These five steps are described (...)'.

As one might expect, arbitrariness, convention, and hence 'subjectivity' are inevitably present in the initial stages, when several choices are made by the developer. The formal method is unable to eliminate subjectivity, but propagates it stealthily throughout the entire process.

Ignoring the subjective character of choices such as those pointed out above, and still evoking the absolute certainty in formal methods, we can see nowadays in the CC web site statements such as: 'IT products and protection profiles which earn a Common Criteria certificate can be procured or used without the need for further evaluation' suggesting that formal methods are reliable enough to bypass the need of any additional testing, and overshadowing that its role is to provide strong evidence that the system does not contain critical errors.

III. When objectivity is no longer enough

In software verification and validation, the criterion of truthfulness, reliability and applicability is many times dependent upon confidence in proofs, which, in turn, has been historically linked to the purely deductive reasoning (or 'the ideal of certainty achieved by mathematical proof', in words of Hoare [16]). This does sound a bit contradictory since, in computer science, the abstract (formal) knowledge becomes directly embodied in computer programs, and so, apparently makes direct contact (without intermediation) with the 'life-world', that is, borrowing the term from Edmund Husserl, the 'only real world, the one that is actually given through perception, that is ever experienced and experienceable - our every-day life-world' [17]. Thus, inductive reasoning, tests, empirical approaches as well as methodologies based on social collaboration appear in programming activities side by side with formal methods, as a way of approaching 'our every-day life-world'. Computer programming evinces that knowledge is a situated construction, that is, a construction strongly connected to materialities and local issues, even when the subject is mathematics or other technical or abstract subjects.

Let us consider the controversies around the establishment of the IEEE Standard for Floating-Point Arithmetic [18]. It shows the conflict between the 'objective' arithmetic and the requirements of 'every-day life-world', here embodied in a hardware architecture. In this struggle, both sides change, giving rise not only to a modified computer but, as a counterpoint of Frege's claim ('there is nothing more objective than the laws of arithmetic' [6]) giving rise also to a modified arithmetic.

The core issue was the confrontation of the infinite expansion of certain real numbers and the finite size of computational representation, which certainly requires some form of truncation. Different algorithms were used by different companies (IBM, Digital, HP, Intel, Texas) which generated different results for the same purpose. A comparison between them showed that there were many decisions to be taken: 'One specialist cite[d] a compound-interest problem producing four different answers when done on calculators of four different types: $331,667.00, $293,539.16, $334,858.18 and $331,559.38. He identifie[d] machines on which a/1 is not equal to a (as, in human arithmetic, it always should be) and $e\pi - \pi e$ is not zero.' [18]

The total number of digits to be adopted in the computational representation of real numbers was a decision that involved the complexity of the hardware being used. In addition, other decisions would also influence hardware design, such as the size of the sequence of digits for representing the mantissa and exponent in the floating point representation. Mackenzie [18] also pointed out the mathematical arbitrariness embedded in several choices: what should be done if the result of a calculation exceeds the largest absolute value expressible in the chosen system, or if it falls below the lowest? What should be done if a user attempts a meaningless arithmetic operation such as dividing zero by zero? In addition to producing different results in some calculations, the lack of standardization hindered compatibility among different computers. Thus, it was necessary to define a standard computational arithmetic.

 ISBN: 978-1-61208-233-2

However, changes in the numerical representation implied costly hardware changes and other nuisances such as lack of compatibility with preexisting programs. Fundamental questions persisted: these different forms of representation configured a new arithmetic or were they just different ways of representing the sole real arithmetic?

'Negotiating arithmetic', as Mackenzie aptly termed it, proved to be a long process. A committee began to work in 1977 but the norm IEEE 754, Numbers Fractional Binary Arithmetic, was not adopted until 1985. The crucial point, highlighted by [18], is that 'there was a stable, consensual human arithmetic against which computer arithmetic could be judged. Human arithmetic was, however, insufficient to *determine* the best form of computer arithmetic. (…) Human arithmetic provided a resource, drawn on differently by different participants, rather than a set of rules that could simply be applied in computer arithmetic.'

IV. Social Processes for Software Correctness

Despite the strength of mathematization of software, even in the seventies the confidence in formal methods was not a consensus: '[L]et us suppose that the programmer gets the message 'VERIFIED.' (. . .) What does the programmer know? He knows that his program is formally, logically, provably, certifiably correct. He does not know, however, to what extent it is reliable, dependable, trustworthy, safe; he does not know within what limits it will work; he does not know what happens when it exceeds those limits. And yet he has that mystical stamp of approval: "VERIFIED." ' [19]. Hence, subjectivity was clearly pointed out, but was insufficient to shake the confidence that rested solely on formal methods, and even applies today.

The dissenting voices did more than point out the existence of a social component in the acceptance of theorems and proofs. They argued that it is precisely the social component that acts as a decisive factor of trust and may lead to minimize the error conditions: 'What elements could contribute to making programming more like engineering and mathematics? One mechanism that can be exploited is the creation of general structures whose specific instances become more reliable as the reliability of the general structure increases. This notion has appeared in several incarnations, of which Knuth's insistence on creating and understanding generally useful algorithms is one of the most important and encouraging. Baker's team-programming methodology is an explicit attempt to expose software to social processes. If reusability becomes a criterion for effective design, a wider and wider community will examine the most common programming tools.'[19].

Although these ideas have not strongly echoed then, we now see that 'expose software to social processes' seems to be the trend in the development of secure software. Researchers cited generally useful algorithms that took the form of the present design patterns which are general descriptions of how to solve a commonly occurring problem in software design. A pattern is an unfinished algorithm that must to be adapted to many real situations. They also cited team programming methodologies that nowadays have been improved by the collaborative capabilities introduced through the Internet, in a way that a piece of code can be constructed or examined by several hands, tending to stability. Reusability is a key issue in the conception of modern program environments, enabling stable codes to be used as components in the construction of modules. Finally, there is currently a proliferation of software development methodologies which rely on social collaboration for secure software development. Some examples are TDD, PBL, social coding, pair programming. Test Driven Development (TDD) is a programming methodology where any functionality of a program starts from a failing test case. Each piece of code is written to solve the test case and overpass its failure. Problem Based Learning (PBL) is a learning methodology that considers a realistic problem, with all its complexity, as a way of invoking interdisciplinarity and autonomy aiming at knowledge construction through the design and implementation of a solution for the proposed problem. Both TDD and PBL takes place in teams. Social Coding are face events aiming the development or enhancement of code in groups. Pair Programming is a programming mechanism guided by a "pilot" trading ideas with a "co-pilot" attended by an audience. Each of these methodologies, among others, start from the assumption that the collective creation, negotiation, discussion and review by multiple agents, among other mechanisms of participation, tend to maximize the chances of success in building a product, especially software.

V. Conclusion

In the Strong Program in the Sociology of Knowledge of the University of Edimburgh, case studies play an important role as they bring in the complexity of the 'life-world' situations. The Sociology of Mathematics, a sub-area of the Strong Program, is a field were the resistance against a intertwined approach to mathematics is a key issue of study, and case studies make visible this resistance. David Bloor, one of the proponents of the Strong Program, referring to questions involving the myth of a purified mathematics, claimed that '[t]he best answer to these questions is to provide examples of such sociological analyses' [12].

In the line of the Sociology of Mathematics, this article pointed out that strategies of software certification currently in place and the tone of recent initiatives such as 'The Great Challenge' indicate that confidence in purified mathematics still underlies the thinking and the doing of mathematicians and computer scientists. In the sequence, this article brings in a case study concerning the definition of an arithmetic for computers that makes a compelling argument about the impossibility of achieving a purified arithmetic, that is, an arithmetic that is not influenced by what is considered 'extra-mathematical' factors. In a attempt to have purified mathematics as an arbiter, new elements and testimonies have slowly emerged, destabilizing the bases of the search for objectivity and making room for mixed heterogeneous (extra mathematical) elements that were decisive in establishing consensus. We thus conclude this paper by citing two recent cases that argue in favor of hybrid approaches, rejecting the possibility of a mathematics that, being supposedly free of subjectivities, would provide absolute certainty.

 ISBN: 978-1-61208-233-2

The first case is a recent discussion referring to a famous phrase of the mathematician Georg Cantor: '*Je vois mais je ne le crois pas*', by which he would have expressed his astonishment at the amazing results that he had just discovered. According to the analyses of the mathematician Gouvêa [20], however, this phrase was actually a response to Dedekind who argued contrary to Cantor's proposals. It was an emphatic trope against his opponent's arguments about something that was, for Cantor, completely clear. Gouvêa's conclusions about this case have much to do with the discussion of objectivity and the influences of non-mathematical factors in the configuration of what is said to be objective. According to Gouvêa (in page 198) '[t]he story was then co-opted to demonstrate that mathematicians often discover things that they did not expect or prove things that they did not actually want to prove.'

Sociology of Mathematics argues that mathematical knowledge is a result of several steps of agreement within a collective thought, in strong alignment with Gouvêa's assertion about subjectivity in mathematical proofs: 'A proof is not a proof until some reader, preferably a competent one, says it is. Until then we may see, but we should not believe.'

The second case is about a recent statement of Tony Hoare, a well known knowledgeable spokesman, for the use of formal methods to ensure program correctness. As late as 2010 Hoare felt adequate to announce a reconsideration of his own previous words. In page 5 of [21]: 'I regarded program testing as the main rival technology'. He reported a join work where he could see senior researches using formal methods not for proof but to detect program errors as much close as possible to their occurrence in code. He then concludes: 'Testing and proving are not rivals: they are just two ends of a scale of techniques available to the software engineer to collect evidence for the validity and serviceability of delivered code.' Hoare's testimony in a year as recent as 2010 shows for how long inductive reasoning, tests, empirical approaches have been (and very likely still are) rejected as legitimate mechanisms for software verification.

The Sociology of Mathematics bring legitimacy to explanations of mathematical facts (such as proved theorems) which distance themselves from explanations of a more absolutist flavor prevailing among the majority of mathematicians. For the Social Studies of Science and Technology, where the universality of knowledge is understood as a mechanism to ensure authority and science is viewed as a local phenomenon, objectivity is addressed in its interweaving with the social; this makes it possible that other elements besides those considered as 'technical' come into play in the composition of the facts regarded as 'mathematical'. A closer examination of mathematical practice shows this inevitable interweaving of knowledge, which, however, remained invisible to the great majority possibly because of the lack of interest in shaking the stability of a purified body of knowledge which places mathematics in a level of unquestionable, neutral and universal truth.

References

[1] E. W. Dijkstra, "The humble programmer." Commun. ACM, vol 15, issue 10, pp. 859-866, October 1972.

[2] P. Naur and B. Randell, Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, pp. 7-11 October 1968.

[3] T. M. Porter, Trust in numbers: the pursuit of objectivity in science and public life, Princeton University Press, 1995.

[4] D. A. Mackenzie, Mechanizing proof: computing, risk, and trust. Cambridge, Mass. MIT Press. 2001.

[5] J. S. Mill, A System of Logic, Raciocinative and Inductive, Londres, H& B Pub, 1848.

[6] G. Frege, The Foundations of Arithmetic. A logico-mathematical enquiry into the concept of number, Harper & Brothers. New York. 2Ed, 1953.

[7] R. Carnap, O. Neurath, and H. Hahn, "La concepción científica del mundo: el Círculo de Viena. 1929" Trad. Lorenzano, P. Presentación de La concepción científica del mundo: el Círculo de Viena. Revista Redes, vol.9, n. 18, pp. 103-149,2002..

[8] D. Hilbert, "On the infinite", In: Bencerraf, P., Putnam, H. Eds. Philosophy of Mathematics Cambridge University Press, 1984.

[9] M. Davis, The undecidable; basic papers on undecidable propositions, unsolvable problems and computable functions, Raven Press Hewlett, N.Y., 1965.

[10] D. J. Struik, "On the Sociology of Mathematics", Science and Society, New York, vol. VI, no. 1, Winter, 1942.

[11] C.A.R. Hoare and J. Misra, "Verified Software: Theories, Tools, Experiments Vision of a Grand Challenge Project". In B. Meyer, J. Woodcook, eds. First IFIP TC 2/wg 2.3, VSTTE 2005, Zurich, vol 4171 of LNCS pp. 1-18, Springer, 2005.

[12] D. Bloor, Knowledge and social imagery. Chicago: University of Chicago Press, 1976/1991.

[13] Department of Defense Standard. Department of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD, Dec, 1985 av. at http://csrc.nist.gov/publications/secpubs/rainbow/std001.txt [retrieved:September,2012]

[14] Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components July 2009 Version 3.1 Revision 3. CCMB- 2009-07-003:229.

[15] C. L. Heitmeyer, "On the role of Formal Methods", Electronic Notes in Theoretical Computer Science, vol 238, pp. 3–9, 2009.

[16] C.A.R. Hoare, "How did software get so reliable without proof?" FME'96: Industrial Benefit and Adv. in Formal Methods, vol. 1051 , pp. 1-17, 1996.

[17] E. Husserl, The crisis of European sciences and transcendental phenomenology; an introduction to phenomenological philosophy. Evanston: Northwestern University Press, 1954/1970.

[18] D. A. Mackenzie, Negotiating Arithmetic, Constructing Proof. In: D. Mackenzie Ed. Knowing Machines - Essays on Technical Change. Cambridge, MA: The MIT Press, 1996.

[19] R. A. De Millo, R. J. Lipton, and A. J. Perlis, Social Processes and Proofs of Theorems and Programs, Commun. ACM vol 22, pp. 271-280, 1979.

[20] F. Q. Gouvêa, "Was Cantor surprised?", The Mathematical Association of America, Monthly vol 118, pp. 198-209, 2011.

[21] C. A. R. Hoare. "Testing and proving, hand-in-hand" TAIC PART'10, L. Bottaci and G. Fraser Eds. Springer-Verlag, Berlin, Heidelberg, pp. 5-6, 2010.

 ISBN: 978-1-61208-233-2