Software Testing and Formal Specifications: a turbulent history

Marie-Claude Gaudel

Honorary Professor, LRI, Université Paris-Sud, France
marieclaude.gaudel@gmail.com

Extended Abstract[1]

Software testing based on formal specifications[2] has a slightly troubled history with, let us say, a moderate happy end. This talk will report on the problems encountered by the first researchers in the field. Then, it will attempt to analyse how and why this approach of software testing has been recognized by the two concerned sub disciplines of computer science, namely software engineering and formal verification of software.

A bit of history.
In the seventies, most actors of both fields considered that they had nothing to bring to each other. Even worse, some influential scientists from both sides emitted mutual doubts on the pertinence of these fields: from the side of the software engineering research community, one can cite De Millo et al. [5], where the interest and feasibility of the formal verification of programs is fiercely negated; and from the side on the formal approaches to programming, one can cite the famous Dijkstra curse against testing [6], which definitely deserves to be quoted:

> *Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness*

Dijkstra was wrong[3]. He mixed up the notions of programs and systems.

A system is a dynamic entity, embedded in the physical world. It is in essence quite different from a piece of text (formula, program) or a diagram.

A program text, or a specification, or a model, are not the system but some description of the system. It is possible to reason about it, to prove some property, as the work of Dijkstra and many others have made possible, but those properties don't take into account the execution environment and its hazards.

The map is not the territory [13].

[1] Some parts of this abstract are customised excerpts of [8]

[2] According to Wikipedia: "These specifications are formal in the sense that they have a syntax, their semantics fall within one domain, and they are able to be used to infer useful information".

[3] as well were De Millo et al.: nowadays, formal specifications and formal verifications of programs are possible and bring much to systems reliability.

Actually, this debate reflected the gap between two communities: the software engineering researchers, mainly aiming at practical applications, working in majority in Northern America; the more academic computer scientists, studying programming languages semantics, working in majority in Europe. In some sense, this topic, test generation from formal specifications, was a risky attempt to bring ideas across a community divide.

The way towards formal testing.
Note that the debate was between formal proofs and testing. Formal specifications were studied as bases for program proofs, exclusively. The possibility that formal specifications could be used for generating test inputs was not even envisaged during this debate. In such a context, claiming that it was worthy to explore required some bravery.

At the beginning of the eighties, the idea of deriving tests by instantiating the axioms of algebraic specifications was presented in Luc Bougé thesis [3] under my direction and independently at the same time by John Gannon et al. [7] in the US, and Pankaj Jalote [12] in India. Publishing this results was far from obvious for these three pieces of work. In our case, we succeeded finally, both in some formalist venue [4], and in some software engineering journal [2], thanks to the support of a few colleagues and friends that believed in this approach.

Nowadays, this approach has been applied to a variety of formal specification methods. See for instance [10] publised in 2009. Almost every new formal approach comes with a method for test derivation. Why has it been successful[4] at last?

Perhaps because it was an excellent idea... But we know it is not a realistic hypothesis. There are a few more plausible explanations.

A first one is that in addition to the intrinsic interest of writing formal specifications, it provides some justifications that are interesting for software developers who often are reluctant to use them: designing efficient test sets is time consuming and difficult. The fact that, thanks to their formal definitions, some tools can be developed to generate them from such specifications makes them more attractive. On the other side, the academic community was pleased with this argument for the applicability of their work. Developing such ideas could be seen as a win-win research operation.

Another one is that at the same time another research community had been very active in the domain with very strong motivations and a significant economic weight: the designers of telecommunication protocols (see [14] among the early publications). Such protocols must be agreed upon by various entities such as international agencies, operators, and development firms. As these firms generally don't disclose their source code, technical standards have been stated for specification and conformance testing [11], which is the basis of their certification. These formal specification languages were oriented to temporal ordering and synchronisations. In contrast, the formal approaches mentioned above mainly took into account treatments of values with complex data types. This

[4] Actually, it is not a full success: there are still some pockets of resistance in both camps.

complementarity led to common works ([1, 9] for instance) and strengthened the interest in formal testing of both communities and most computer scientists.

Similarly, chips designers had been developing for quite a while test methods and tools based on logical descriptions of such circuits (there is a good introduction in [15]). Thus some senior researchers in this area found natural to base software testing on formal specification and supported the idea.

Some lessons learned.

Anyway, fighting conformism is not an easy task and there is no silver bullet to lead and win such a fight. I mention below a few points that helped in my case.

First, one needs to be strongly convinced and obstinate, ready to ignore negative positions and discourses even from big names. As reported above, it is useful to exchange with scientists of related but different topics, and to convince some big names among them.

Locally, one must be able to find free time to work on it, some PhD students, some equipments, and fees for attending scientific events. Open-minded management was important. Moreover, obtaining funding required a touch of insecurity, I dare to say. Applications must not attack head-on mainstream ideas. Of course, what is promised in such applications must be done, but reviewers never complain if in addition some interesting byproduct arises.

References

1. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Comput. Networks **14**, 25–59 (1987)
2. Bougé, L., N.Choquet, Fribourg, L., Gaudel, M.C.: Test set generation from algebraic specifications using logic programming. Journal of Systems and Software **6**(4), 343–360 (1986)
3. Bougé, L.: Modeling the notion of program testing; application to test set generation. Thesis, Université Pierre et Marie Curie - Paris VI (Oct 1982), https://tel.archives-ouvertes.fr/tel-00416558, directeur: Prof. Marie-Claude Gaudel
4. Bougé, L., Choquet, N., Fribourg, L., Gaudel, M.C.: Application of Prolog to test sets generation from algebraic specifications. In: International Joint Conference on Theory and Practice of Software Development (TAPSOFT). LNCS, vol. 186, pp. 261–275. Springer (1985)
5. DeMillo, R.A., Lipton, R.J., Perlis, A.J.: Social processes and proofs of theorems and programs. Commun. ACM **22**(5), 271–280 (1979)
6. Dijkstra, E.W.: The humble programmer. Commun. ACM **15**(10), 859–866 (1972)
7. Gannon, J.D., McMullin, P.R., Hamlet, R.G.: Data-abstraction implementation, specification, and testing. ACM Trans. Program. Lang. Syst. **3**(3), 211–223 (1981)
8. Gaudel, M.C.: Formal specifications and software testing, a fruitful convergence. In: International Workshops. FM 2019. Lecture Notes in Computer Science, vol. 12233. Springer Verlag (1995), https://doi.org/10.1007/978-3-030-54997-8_5
9. Gaudel, M.C., James, P.R.: Testing algebraic data types and processes: A unifying theory. Formal Asp. Comput. **10**(5-6), 436–451 (1998)
10. Hierons, R.M., Bogdanov, K., Bowen, J.P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P.J., Lüttgen, G., Simons, A.J.H., Vilkomir, S.A., Woodward, M.R., Zedan, H.: Using formal specifications to support testing. ACM Comput. Surv. **41**(2), 9:1–9:76 (2009)
11. ISO: Conformance testing methodology and framework. International Standard is-9646 (1991)
12. Jalote, P.: Specification and testing of abstract data types. In: IEEE International Computer Software and Applications Conference COMPSAC. pp. 508–511 (1983)
13. Korzybski, A.: Science and Sanity: A Non-Aristotelian System and its Necessity for Rigour in Mathematics and Physics. Institute of General Semantics (1933)
14. Sarikaya, B., von Bochmann, G.: Some experience with test sequence generation for protocols. In: Second International Workshop on Protocol Specification, Testing and Verification. pp. 555–567. North-Holland (1982)
15. Wang, L.T., Wu, C.W., Wen, X.: VLSI test principles and Architectures. Morgan Kaufmann (2006)