STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

WHY MUST WE TEACH VERIFICATION AND VALIDATION TO UNDERGRADUATES?

MILITON FRENŢIU AND FLORIN CRĂCIUN

Abstract. The way program verification is taught is firstly described. The verification methods are shortly presented and the objectives of the Verification and Validation (V&V) course are discussed. The main gain on the students education is finally underlined.

1. Introduction

The most important property of a program is whether it accomplishes the intentions of its user, i.e. if it is correct. We have all observed the permanent and rapid growth of computer usage in all fields of human activity. The need for new programs is immense and their complexity is continuously growing. There are known programs with millions of lines written by hundreds of people. These more complex programs cannot be developed like the old small ones. It has become necessary to analyse software costs over the entire life cycle of the system. It is known today that the major errors are due to poor design of the system not to bad coding. After all, the fraction of time needed for programming is about 20% of the entire development process.

The need for more reliable software [20, 25, 14], and the role of Verification and Validation [3, 15, 35] are well known. Investigations have shown that formal verification procedures would have revealed the exposed defect in, e. g., the Ariane-5 missile, Mars Pathfinder, Intels Pentium II processor, or the Therac-25 therapy radiation machine [5].

Today software systems have become essential parts of our daily life. Computers are used in all fields of human activities. More and more programs are needed, and their complexity increases continuously. Software critical systems [1] require error-free programming. To obtain programs without errors we

Received by the editors: April 8, 2011.

2010 *Mathematics Subject Classification.* 68Q60, 68N30.

1998 *CR Categories and Descriptors.* D.2.4 [**Software**]: Software engineering – *Software/Program Verification*; D.2.5 [**Software**]: Software engineering – *Testing and Debugging*.

Key words and phrases. verification and validation, education.

need a new way of writing programs, new people, much better educated, able to do this.

There is a contradiction between the desire to obtain a system as quickly as possible, and to have a correct system. The experience shows that more than 75% of finished software products have errors during maintenance, and deadlines are missed and the cost overrunning is a rule, not an exception. It was estimated [34] that more than 50% of the development effort is spent on testing and debugging. Nevertheless, some errors are not detected by testing, and some of them are never detected. More, there are projects that have never been finished [8]. And it is not an exception; it is estimated that from each six large projects two of them are never finished.

Almost all students have learned some programming at school. They think they know what programming is all about. They express resentment when they are forced to document their activity, to design the program, or to think to the correctness of their programs. They go directly to computer and introduce their programs, and then run them. If the first execution seems OK they are satisfied. They don't like to think first to the algorithm and to describe it on the paper. Also, they are not used to test seriously their programs. We need to fight with these people, to change their habit, to educate them in a different way. That is why we have to stress from the very beginning that "programming is a high intellectual activity" [19], that "they must think first, program later" [23], that the correctness is the most important property of good programs.

2. The contents of Verification and Validation Course (V&V)

The validation and verification of software systems is a major issue in Software Engineering. The objective of this course is to train future computer scientists and engineers in the fundamental concepts on which verification and validation of software systems are based. Though the orientation of the course is practical in nature, its goal is to teach the fundamental principles that are needed to build reliable systems needed in various fields.

V&V is one of the important courses that contribute to obtain well-educated practitioners. We teach such a course to the third year undergraduates [15]. The theoretical basis for building reliable software products is given here. The course consists of three main parts:

- the theory of program correctness;
- the methods of verification and validation;
- the consequences on software engineering practice.

The entire curricula of this course, and also, the undergraduate study program may be seen at [31].

One cannot understand V&V if she/he does not know the concept of program correctness. We are confident that we cannot prove the correctness of a large program with one million lines. But, as Dijkstra [4] underlined, it is necessary to use perfect (correct) simple algorithms for building reliable large programs (but it is not sufficient). The first part of the course presents this concept and gives methods to prove correctness. The Floyd and Hoare methods [10, 18] to prove correctness are given. Then, more important, the accent is put on the methods to achieve this correctness by refinement from specifications. Also, the roles of Dijkstra [6], Hoare [18], Gries [17], Droomey [7], and Morgan [27] are mentioned.

The verification methods discussed in the second part are: proving correctness (already mentioned), testing, inspection, symbolic execution, and model checking. Testing is the oldest verification method and often the only one used. Symbolic execution [22] although similar to testing, differs from testing by the values given to the input variables, which are symbols not concrete values.

Inspection, introduced by Fagan [9], with all later variants (peer reviews [33], walkthroughs [30], active design reviews [29]) is a newer V&V method, as well as Model Checking [2, 21]. Inspections can find up to 80% of defects, much earlier than testing. The formal process developed by Fagan was improved by Gilb [16], who considers that the main advantage of inspection over testing is its possibility of process improvement. The feedback obtained from inspections is useful to eliminate defects and bad habits from the development processes. Also, the cost of eliminating an error by testing is 14.5 times higher than the cost of eliminating it by inspection [32].

At the undergraduate level we cannot afford to teach the mathematical basis of model checking, the theory that lies at the basis of model checking. Instead, the main theoretical aspects are presented in a two hours lecture, and the existence of tools and examples of such tools are shortly presented [15].

It is underlined that all of these methods must be practiced during the software process [15]. Their usage may be informal, or more formal, complete or only some of them, depending on the type of the system which is built. For safety-critical systems all of the above mentioned methods must be used. We consider that all future software engineers should be aware of all verification methods.

The third part of the course presents the Cleanroom methodology [26, 24], the role of V&V for Software Quality Assurance and Software Process Improvement, and the consequences of correctness theory on software engineering practice [11, 14, 15].

3. Consequences of teaching Verification on Software Engineer

We think that we educate our students for their future profession, that last for several decades. Also, they must be prepared for changes in software engineering. They need to acquire now the knowledge needed to build more reliable systems. Certainly, proving correctness of algorithms is not enough to obtain more reliable systems, but it is necessary. For years we ask the students to understand and to respect some important rules of programming [11, 13, 14, 23]. Many of them are simple consequences of the theory of program correctness [11].

The fact that program verification is a very important activity which extends from the first statement of the problem, until the end of the project is repeated many times. And a special attention was given to the inspection of all documents [9, 28]. The students must hear from the beginning that testing is not enough, that inspection of all phases may be more useful.

The future software tester must be well prepared to test software products, the member of the verification team must be an expert in verification activities. Not all students will work in a verification team, but all of them must contribute to build reliable systems.

So, why must we teach Verification and Validation? The knowledge acquired teaching a V&V course have serious consequences on the entire activity of a software engineer. These consequences may be extracted as important rules that must be obeyed by all participants in the software process. We mentioned here only few of them:

- Prevention is better than cure!
- Think first, program later!
- Do it right the first time!
- Prove the correctness of your algorithms!
- Write documentation for all software activities!
- Inspection is superior to testing!
- Discover and Remove the errors as early as possible!
- Use comments to document your code. Use assertions.
- Pay attention to the clarity of your software documents!
- Respect the standards!

4. Conclusions

We need confidence in the quality of our software products. We need to educate the future software developers in the spirit of producing correct, reliable systems. For this we must teach students to develop correct programs. We are aware that usually programmers do not prove the correctness of their programs. There always has to be a balance between cost and the importance

of reliability of the programs. But just when the well educated people do not prove the correctness, their products are more reliable than the products of those "programmers" who never studied program correctness. Therefore, we consider that the students must hear, and must pay attention to the correctness of their products.

From first year, students are taught about program specification and design. The entire life-cycle is presented, the importance of documentation for all steps is underlined. Top-down and the other programming methods are taught, and the students hear that the design is more important than coding. Each part of the design must be specified, the code must be explained by comments, the comments should be neither more nor less than needed. Since we also observed that students do not like to write comments [12] we tried to explain their necessity, and to force them to document the code [13].

References

1. R. W. Butler and S. C. Johnson, *Formal Methods For Life-Critical Software*, Computing in Aerospace 9 Conference (San Diego, California), 1993, pp. 319–329.
2. E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.
3. E. M. Clarke and J. M. Wing, *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys **28** (1996), no. 4, 626–643.
4. O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*, American Press, 1972.
5. N. Dershowitz, *Software Horror Stories*, www.cs.tau.ac.il/∼nachumd/horror.html.
6. E. W. Dijkstra, *Guarded Commands, Nondeterminacy and Formal Derivation of Programs*, Communications of the ACM **18** (1975), no. 8, 453–457.
7. G. Dromey, *Program Derivation. The Development of Programs from Specifications*, Addison Wesley, 1995.
8. Oz Effy, *When Professional Standards are LAX. The CONFIRM Failure and Its Lessons*, Communications of the ACM **37** (1994), no. 10, 29–36.
9. M. Fagan, *Design and Code Inspections to Reduce Errors in Program Development*, IBM Systems Journal **15** (1976), no. 3, 182–211.
10. R. W. Floyd, *Assigning Meanings to Programs*, Mathematical aspects to computer science. Proc.Symposium in Applied Mathematics (J. T. Schwartz, ed.), no. 19, American Math. Soc., 1967, pp. 19–32.
11. M. Frentiu, *On Program Correctness and Teaching Programming*, Computer Science Journal of Moldova **5** (1997), no. 3, 250–260.
12. ———, *The Impact of Style on Program Comprehensibility*, Proceedings of the Symposium Zilele Academice Clujene (2002), 7–12.
13. ———, *On programming style*, www.cs.ubbcluj.ro/∼mfrentiu/articole/ style.html, 2003.
14. ———, *Correctness, A Very Important Quality Factor in Programming*, Studia Univ. Babeş-Bolyai, Seria Informatica **L** (2005), no. 1, 12–21.
15. ———, *Verificarea si Validarea Sistemelor Soft*, Ed. Presa Universitara Clujeana, Cluj-Napoca, 2010.
16. T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley, 1993.
17. D. Gries, *The Science of Programming*, Springer Verlag, 1981.

18. C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Communications of the ACM **12** (1969), no. 10, 576–580.
19. ———, *The mathematics of programming*, Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, 1985, pp. 1–18.
20. C. M. Holloway, *Why Engineers Should Consider Formal Methods*, 16th AIAA/IEEE Digital Avionics Systems Conference, vol. 1, 1997, pp. 1.3–16–1.3–22.
21. J. P. Katoen, *Principles of Model Checking*, MIT Press, 2008.
22. J. C. King, *Symbolic Execution and Program Testing*, Communications of the ACM **19** (1976), no. 7, 385–394.
23. H. F. Ledgard, *Programming Proverbs for Fortran Programmers*, Hayden Book Company Inc., New Jersey, 1975.
24. R. C. Linger, *Cleanroom Software Engineering for Zero-Defect Software*, 5th Intern. Software Engineering Conference (Baltimore), IEEE Comp. Soc. Press, 1993, pp. 1–13.
25. B. Meyer, *Software Engineering in the Academy*, IEEE Computer **34** (2001), 28–35.
26. H. Mills, M. Dyer, and R. Linger, *Cleanroom Software Engineering*, IEEE Software **4** (1987), no. 5, 19–25.
27. C. Morgan, *Programming from Specifications*, Springer, 1990.
28. A. Myers, *A Controlled Experiment in Program Testing and Code Walkthroughs Inspection*, Communications of the ACM **21** (1978), no. 9, 760–768.
29. D. L. Parnas and D. M. Weiss, *Active Design Reviews: Principles and Practices*, 8th Intern. Conf. on Software Engineering, 1985, pp. 215–222.
30. S. R. Schach, *Software Engineering*, IRWIN, Boston, 1990.
31. UBB, *Undergraduate Study Program, Babeş-Bolyai University*, http://www.cs.ubbcluj.ro, 2011.
32. K. E. Wiegers, *Improving Quality Through Software Inspections*, Software Development **3** (1995), no. 4, 55–64.
33. ———, *Seven Truths About Peer Reviews*, Cutler IT Journal (2002).
34. E. Yourdon, *Modern Software Analysis*, Yourdon Press, Prentice Hall Building, New Jersey, 1989.
35. M. V. Zelkowitz, *Role of Verification in the Software Specification Process*, Advances in Computers, no. 36, Academic Press, 1993, pp. 43–109.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania

E-mail address: mfrentiu@cs.ubbcluj.ro, craciunf@gmail.com