© 2014 IJIRT | Volume 1 Issue 11 | ISSN: 2349-6002

Software Testing Research and Software Engineering

Amrendra Kumar Upadhyay, Ankit Bhatt, Anil Pilaniya
Dronacharya College Of Engineering, Gurgaon
Sec-5, Haryana-122506

***Abstract*- Software testing research has not kept up with modern software system designs and applications, and software engineering education falls short of providing students with the type of knowledge and training that other engineering specialties require. Testing researchers should pay more attention to areas that are currently relevant for practicing software developers, such as embedded systems, mobile devices, safety-critical systems and other modern paradigms, in order to provide usable results and techniques for practitioners. We identify a number of skills that every software engineering student and faculty should have learned, and also propose that education for future software engineers should include significant exposure to real systems, preferably through hands-on training via internships at software producing firms.**

***Index Terms*- Reliability, Software Testing, Software Engineering.**

I. ISSUE IN SOFTWARE TESTING RESEARCH

Where is software going - all those billions or trillions of lines of code currently running and the gazillions more that will be written in the next decade and how does it relate to the current software engineering research literature? Where the research community is headed and are research and practice converging? When we write our research papers, is there anyone out there listening or are we writing for ourselves and for each other?

The sorts of software systems discussed in the software testing research literature, by and large, are systems that are either stand-alone, or that connect with other software systems that run on what are typically thought of as computers. These systems take inputs which are characters, or numbers, or files of characters and numbers. It is relatively easy to understand how to test them, even if it is not done very well, or very thoroughly, or if good ways of assessing the comprehensiveness of the tests are lacking. Typically in the research community, testing is equated with *functionality* testing. The sorts of issues that are addressed are how to generate and select test cases, how to do it efficiently, how to assess adequacy, etc.

Of course, all of these are important issues, but this research has been done for decades and very few of its results have changed the way software is tested in any fundamental way. We believe this is because researchers are not talking about the types of software that industry and government are increasingly concerned about, and are not talking about testing for the types of problems that are of the greatest concern for these systems. Additionally, researchers generally do not provide compelling evidence that the techniques they propose in their research will actually be successful or be practically beneficial. Finally, practitioners often complain about the lack of robust tool support for a proposed testing research approach. If a prototype tool that is hard-to-use and understand is provided by the researchers, practitioners will be very reluctant to spend time learning it, especially when the benefits are doubtful, and its operation is frustrating. If the task of building a usable tool is left to its potential users, it will almost certainly not happen. Practitioners have their hands full with the subject system they are building; they are generally not willing to invest significant time out of their already overstretched schedules to implement a new technique that they view as unproven because there are no large-scale empirical studies to back it up.

Furthermore, these sorts of systems of systems are by no means unique to the military or to the automotive industry. Embedded systems are in every industry, and they are increasingly driven by analog inputs such as pulses, or electrical inputs, or a continuously variable mechanical action, all of which are far removed from anything the end-user is aware of. For example, one might have to test an automobile fuel injection software system, which responds to another system that reacts to a driver’s depressing a gas pedal.

Testing researchers first have to learn how to test these embedded systems for functionality, even if the system under test is a flight control system for an airplane that is still under design, or a satellite yet to be built. How can one test the functionality of an implanted device that emits a signal or injects some medication into a patient's bloodstream when certain conditions occur, provided that other conditions have not occurred? Once the functional testing has been completed, how can one assure the airplane manufacturer or the satellite designer that the embedded systems are not vulnerable to attack, that they work under all sorts of environmental conditions, that they work when inputs are outside the expected ranges, and that they can meet performance goals, safety regulations and reliability requirements? This is where the research community needs to be headed because this is where the world is heading. And clearly the research community should be arriving ahead of the systems that are being built in industry. Research should be guiding development, but in software engineering, and particularly software testing, that is often not the case.

II. EDUCATION, TRAINING, EXPERIENCE

This section describes what we believe to be the three most important factors in raising the level of software quality and producing a future generation of qualified software engineers. Advances in design, implementation, and validation research are obviously important, but none of them will be ultimately useful without well-trained practitioners who know how to distinguish good design from bad, and who can make intelligent choices of appropriate implementation and validation techniques. The elements of software engineering education include at least the following:

- solid grounding in fundamentals of computer science, including appropriate mathematics
- the importance of working in teams, and how to take advantage of different team members' skills and expertise
- Understanding of all the key factors that might be relevant for a system, when each is appropriate, and how to evaluate them. These factors include such things as
  - risk
  - safety
  - performance
  - reliability
  - correctness (and this might not be the most important)
  - ease of use, clarity

In many engineering disciplines, it is usual for students to have internships which are essentially apprenticeships, where they learn by working with experienced professional engineers and get real hands-on training. Such programs frequently extend an undergraduate engineering degree from four to five years. In many fields, engineering graduates cannot legally call themselves an engineer without passing a licensing exam, and that often has a work experience requirement. For example, it's not enough to know the theory of building a bridge if you want to be a civil engineer; you also have to work with people who design and build them and are experienced enough to mentor interns.

In the United States, these sorts of internships are not the norm in software engineering, and an exam is generally not required for someone to call himself or herself a software engineer. It is not clear that there are any requirements at all that go with the title.

Therefore, it's important to consider how to assure that our software engineering faculty are qualified to actually teach more than foundational courses in the field. One possible solution is for funding agencies to offer summer or even year-long positions for software engineering faculty to work at industrial development and testing organizations. The companies will probably gain very little immediate, concrete benefit from such visitors, and that is why funding agencies should underwrite their expenses. We are *not* speaking about a professor spending the summer or a sabbatical working in an industry research lab - that seldom involves really learning how practitioners specify, design, build or test software, since in many industry labs, researchers are just as far removed from practitioners as academics are.

III. THE BIG PICTURE AND HOW TO GET THERE

In the future we will see more and more embedded software systems, increasingly larger systems of systems, systems that require synchronization with other systems, systems of mobile devices, and safety-critical systems that control all sorts of medical devices and procedures. Since these systems are embedded and depend on other systems, and do not run on devices that look like computers, and are not necessarily directly responding to stimuli controlled by the end user, new ways of testing them need to be developed. This is a significant research challenge.

In most engineering fields, systems are specified using engineering models, which every engineer of the relevant type has been taught to create and understand. That is definitely not the case with software engineers, and modeling needs to be included as a standard tool or skill that every software engineer routinely learns as part of their education. In addition, since embedded software systems are increasingly common and widespread, software engineers need to learn how to simulate systems.

Simulation is a standard tool in many other engineering disciplines, but it is rarely taught to software engineering students. If you are testing a component of a larger system that has not yet been built, the only alternative might be to test it by doing simulations. Other circumstances under which dynamic testing cannot be done at a particular stage of development include software systems embedded in a device that might have disastrous safety consequences if the software were to fail. This might include things like software embedded in medical devices or airplanes. It might be considered too risky to dynamically test the system until it has been compellingly shown to function properly, and the most compelling evidence might come from simulations. While simulation is not a substitute for significant dynamic testing, it certainly does offer the possibility of providing evidence of potential flaws in the system before the airplane is ready to fly, for example

IV. SUMMARY

Far-sighted individuals have called for more attention to engineering principles and sounder education for software engineers for many years [1, 2, 3, 4, 5]. We have tried to offer some concrete suggestions for how we might improve software engineering education, by identifying a number of skills that every software engineering student and faculty should have learned, as well as hands-on training that they should have had. We have also pointed out the following areas that the research community needs to focus on to meet the demands of the types of systems that are being built today and will increasingly be built in the future.

- testing embedded systems
- testing properties other than functionality, including performance, safety and security
- simulation
- industrial grade empirical studies
- easy-to-use tools that implement testing techniques

REFERNCES

www.youtube.com
www.wikipedia.org
www.google.co.in