Awesome Bug Manifesto: Teaching an Engaging and Inspiring Course on Software Testing (Position Paper)

Natalia Silvis-Cividjian
Computer Science Department
Vrije Universiteit Amsterdam
The Netherlands
Email: n.silvis-cividjian@vu.nl

***Abstract*—Although testing software is paramount to safeguard our digitizing society, students are reluctant to consider a career in the field. A reason could be that dedicated courses on software testing are rare. However, even when such a course exists, students perceive testing as a boring, unrewarding and even dogmatic chore. For more than 10 years, we have been teaching a software testing course at the Vrije Universiteit in Amsterdam. Driven by our belief that an abundant exposure to software bugs makes good testers, we experimented with many ideas to engage students and make them love the topic. The most unorthodox, yet effective interventions we are proud of, were: (1) to scare students by analyzing past, software-related accidents, such as Therac-25 or Boeing 737-MAX; (2) to thrill them using bug-hunting gamification, enabled by the in-house developed VU-BugZoo; (3) to trust them an end-to-end testing of safety-critical software-intensive systems, such as model trains, automatic insulin pumps and even radiotherapy facilities, and (4) to inspire their career, by opening a dialog with test professionals from industry. The result is a mature course, read yearly by 50 computer science graduates, where almost 80% of the participants find the topic interesting and challenging, and 40% consider a future carrier in testing. These positive results make us confident that we found a formula that works. In this position paper, we would like to share our innovative ideas and lessons learned. Also in the future, we will stay committed to educate enthusiastic and responsible software testers.**

***Index Terms*—software testing education, student engagement, fault-injection, safety-critical systems, gamification, embedded software, academia-industry cooperation**

I. Problem statement

Once a dream, now an amazing reality, ubiquitous computing trend [1] is developing at this moment at its full speed. This results in software being present everywhere, not only in laptops and smartwatches, but also in washing machines, cars, planes and trains, medical devices, or retail, banking and voting systems. However, such a rapidly digitizing society needs to be properly safeguarded against all kinds of accidental or malicious threats. This calls for a strong team of creative, enthusiastic and responsible testers who must uncover and annihilate these threats as soon as possible, before the product is released to its end-users. And it is our role as educators to train these testers. One may say that given the urgency of the call, everyone who is expected in his career to be in a way or another responsible for a software-based system (so not only its developers) should know how to systematically test it. However, nothing could be further from the truth. In industry, a lot of software testing still happens ad-hoc, except for some safety-critical sectors such as nuclear, aerospace and medical. In academia, testing concepts are often embedded in programming or software engineering courses, but a dedicated course on software testing is not a regular guest in the CS curricula. Recent surveys and panel discussions show that unfortunately, CS students spend only 0 to 27 hours on testing software [2]. In other, non-CS curricula such as physics, chemistry, mechanical and civil engineering, such courses simply do not exist. Even existing software testing courses struggle with a couple of well-known problems. For example, students find testing less exciting than designing or coding. Therefore, only a small fraction consider a future career in testing (in average 20%). Often, the ones who decide to give it a try, struggle to match the expectations of the software industry, which is then forced to train their fresh CS graduates on the job. Testers often complain about fear of complexity, lack of excitement, low appreciation, low salary, and so on [3], [4], [5], [6], [7], [8], [9], [10].

The fact is that an hostile wind towards software testing is blowing from both industrial and academic testing fronts. So, what can we, software engineering educators, do about it? How to wake up students from their apathy ? How to light a spark? How to show students that testing is not just a boring chore, but a skill for life ? How to win souls for a career in testing? The main question we try to answer in this position paper is *"How to engage students in a software testing class and inspire them to continue working in this fascinating field?"*.

II. Our vision

First of all, let us zoom into the concept of software testing using Fig. 1, where one can see its many dimensions, targeting completely different areas of the process. When people talk about testing, they usually mean unit testing, a process of checking that a unit of code behaves according

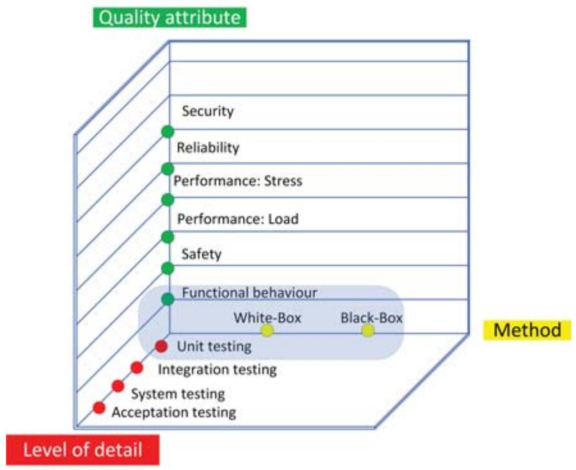


Fig. 1. Different dimensions of software testing. Adapted from [12].

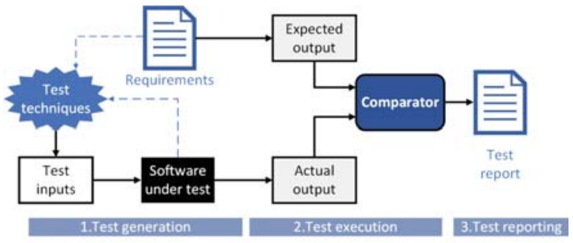


Fig. 2. A generic unit testing process.

to its specifications and/or user intentions [11]. Basically, one executes the code by feeding it with inputs and comparing each result to the expected, specified output (see Fig. 2). If the two are equal, then the test case is declared as passed; otherwise the test is declared as failed, indicating the presence of a defect in code, also called fault, or bug. But there are more levels of testing, such as integration and system testing. And also more quality attributes to test such as performance, or safety.

A lot of efforts have been made to promote software testing as a standalone discipline, resulting in a few solid textbooks that followed the classic, landmark Meyer's "The Art of Software Testing" published in 1979 [13]. As a result, a few dedicated courses on software testing already exist, based on good, solid textbooks [14], [12], [15] that teach mainly functional (black-box and white-box) testing, checking that the system does what it is expected to do. Now one may ask, if good textbooks and dedicated courses are in place, why do software-related accidents still happen?

What we see is that almost all accidents nowadays happen not because a comma was omitted in a line of code, but because sensors do not work, wires break, or people forget, type too fast, panic or do not understand the instructions. For example, in the recent Boeing accidents [16] the MCAS software did exactly what it was intended to do - to push down the aircraft's nose when a stalling was imminent. But was the airplane stalling? No. The problem was a flawed angle-of-attack sensor that made the software believe, wrongly, that it needs to act. Moreover, the pilots were not trained in overruling the MCAS system. The result in both cases was an airplane sent towards the ground and two fatal crashes. If what we hear in the news is true, many mistakes were made during testing this new type of aircraft, but one thing is sure- it was not the software's fault; at least not from a functional point of view. So, maybe it is something missing in the way we teach software testing?

In our opinion, there is a gap between the amount of testing needed in real-life and the testing normally taught in class, limited to the small gray circle in Fig 1. This gap is the main reason for the following problems.

(1) Teaching only unit testing is not enough. Traditional courses do not cover integration- and system testing. This is very unfortunate, because it deprives students of the opportunity to discover interesting problems, such as interface inconsistencies between software modules, or hardware refusing to obey software commands.

(2) Testing only for functionality is not safe. Students often live under the false impression that testing can be stopped as soon as the product provides the intended functionality. However, this is only partly correct. Especially in mission-critical industry, a functional system is not yet ready to be put on the market. It needs a certification that assesses also other system-emergent quality attributes, such as safety and security. Unfortunately, this interesting link between safety analysis and testing is not extensively exploited in traditional courses.

(3) Designing test cases without executing them is not motivating. In our opinion, one of the problems is that we mainly teach students how to design test cases, with no executable in place. A classical assignment in an introductory software testing course usually sounds like : "Generate test cases for requirement X using technique Y". The goal is to develop "a fine nose" on how to choose a test techniques given a certain requirement specification. Which is of course good, but not good enough. The problem is that in this traditional approach, students never reach Step 2 and Step 3 in a testing process, illustrated in Fig. 2. and cannot "catch" the bug at crime scene.

(4) Testing bug-less software is uninspiring. In the rare cases when one provides students with real, executable code implementations, these contain zero to few fascinating bugs. On the one hand, this is normal, if we think that real-life software is developed by competent programmers, who rarely make "stupid" mistakes. On the other hand, this almost perfect artifacts make students perceive testing as a tedious and unrewarding chore.

(5) Testing in an academic-only setting lacks realism. Usually the client in traditional software testing courses are the teachers. They are the ones who eventually perform acceptance testing and grade the students. Unfortunately, this is not the real setting students will find when they will move to industry. There, they will be disappointed to discover that a lot of techniques we preached in class are never being used, mainly because of the challenging trade-off between quality, business success and limited resources.

TABLE I
SOFTWARE TESTING COURSE ORGANIZATION.

Week	Topic	Assignment
1	Basic concepts. Failure analysis. Safety science	FAIL (20h)
2	Black-box testing	PROJ-coder view (15h)
3	White-box testing. Test adequacy.	
4	Integration and System testing	SYS (20h)
5	Automated testing. Special topics	PROJ-tester view (15h)
6-7	Guests	
8	Exam and BUG_HUNT	
9-12		Project systems testing (158h)

At the Vrije Universiteit in Amsterdam, we have been teaching a standalone course on software testing for more than 10 years. During all these years, driven by an almost obsessive fascination for accidents and their causes, we experimented with many innovative ideas, aiming to engage students and make them love the topic [17], [18]. The course is organized as shown in Table I. Additional material such as tutorial and assignments can be found in the course repository [19].

In our course we aimed to improve students' attitude towards testing by addressing all these above mentioned issues. In the next section, we will present a few key innovative ideas that enhanced our course and we will share our findings and lessons learned.

III. A FEW INNOVATIVE IDEAS

A. Learning from mistakes

We deliberately pull out students from their comfort zone at the very start of the course. We want them to realize that unlike programming, testing is not a leisure activity. Or to put it bluntly - if they do not test properly, people will die. Therefore, we start every lecture with a Bug-of-the-day session. Moreover, a first assignment (FAIL in Table I), asks students to analyze notorious software-related failures, including the Therac-25 [20], [21] and Panama radiotherapy accidents, the Patriot missile mishap, the Knight Capital financial disaster, and so on. In doing this, we make use of a novel systems-theory based method for safety analysis, called STAMP [22].

B. Creating end-to-end bug-hunting games

We designed two assignments where students cover all the phases illustrated in Fig. 2. The first is a mini-project (PROJ) that immerses students in a testing experience, both as a developer and as a tester. In the role of developers, students work in groups to specify, implement and test a product such as Hangman game or a heart monitor user interface. Here, students practice unit and integration testing, with the freedom to decide which techniques to use. The code in this project must be developed with the best intentions and is therefore flawless-by-design. Halfway through the project, the students swap their products and start the second, black-box testing phase. Finally, they submit a report with their test strategies and lessons learned. The next assignment is BUG-HUNT, implemented in VU-BugZoo [23] an innovative learning platform, developed in house based on a repository of standalone and embedded code. This time we deliberately-corrupted executable code, to engage students in a fascinating, "test-until-it-explodes" dynamic bug-hunting experience. The idea to use buggy code in teaching software testing is not new [24], [25]. However, our platform distinguishes from similar initiatives mainly through its repository of software embedded in miniature smart homes, autonomous cars and trains, and medical devices. Although the embedded-software repository is still work in progress, the first version of the standalone-software based platform is already functional. It allows to make end-to-end testing challenges and ask students to devise test strategies and describe the bugs they found (see Fig. 3). The tool can be very useful for the whole testing community, especially now, in the middle on the ongoing COVID-19 pandemic, when written exams on the campus are simply not possible.

C. Testing miniature models of safety-critical systems

Both in the SYS assignment and in the follow-up Project Systems Testing we extend testing up to systems level for both safety and functionality. The task is to work in a team to specify, design, code and test a safety-critical autonomous system. What is special here is that students test software embedded in miniature models of real safety-critical systems. This can be an autonomous vehicle, a salinity controlled water plant that mimics an insulin pump, or a H0 scale (1:87.1) miniature railway management system (see Fig. 3). Accidents such as collisions or derailments, overflow and overheating should be avoided at all times. Here students must realize that an exhaustive testing is not possible and 100% safe systems do not exist. What they can realize though in one month is to design a sound, well-justified test strategy that guarantees a well-functioning, "acceptable" safe system that mitigates the hazards with the highest risks.

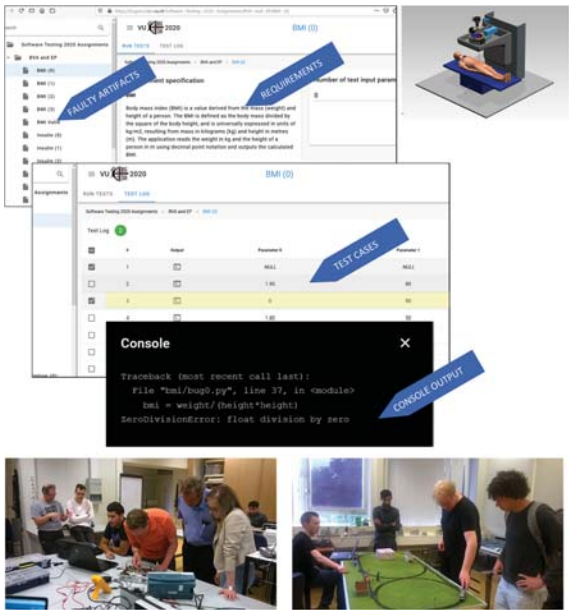
D. Involving industry

Throughout the course, we invite test experts from airlines, railways, and medical industries to present their best practices of organizing the testing process. Also, IT safety experts participate in steering the project and grading the students' test strategies.

IV. EVALUATION

The course is followed yearly by approximately 50 software engineering graduates. From them, 24 students follow the one-month project systems testing. In total, students spend three months and between 150 and 300 hours on software testing, which is much more than reported elsewhere. From the evaluations collected during a 10 years long period (2010-2020), summarized in Table II, one can see that almost 80% of the students constantly found the course interesting and the learning effect adequate.

Students appreciated testing real physical systems, this being the only place where they work with real hardware



```
Traceback (most recent call last):
  File "bmi/bug0.py", line 37, in <module>
    bmi = weight/(height*height)
ZeroDivisionError: float division by zero
```

Fig. 3. An illustration of our innovations, showing the VU-BugZoo screens, a replica of the flawed Therac-25 radiotherapy facility, the railway layout and a view from the lab.

TABLE II
EXCERPT FROM STUDENTS' EVALUATIONS BETWEEN 2010-2020.

Question	-	+-	+	Total	Agree
Interesting course	13	32	155	200	78%
Learned a lot	17	38	140	195	70%
Guest lectures useful	8	11	57	76	75%
Consider a career in testing	13	20	19	52	37%

and embedded software. They discovered that testing only software in isolation is a recipe for accidents. For example, they discovered interesting bugs, situated on the interface between software and hardware. Some of them are similar with the ones which caused real, notorious accidents, such as "command to stop given by software but not executed by the engine, so trains collide", or the more subtle "command to switch given too often which creates switch overheating and melting", "salt added to the system, but sensor not in water, so the software thinks salinity is OK, and does not send command to compensate, which results in an uncontrolled increase in salt concentration, so patient dies", or the very wicked one "corrupted binary transferred, all outputs (wrongly) switch to High, turning on the heater situated on a table, resulting in smoke in the lab".

A first pilot of the newly developed VU-BugZoo used for an online assessment during the COVID-19 pandemic was also very successful. Around 80% of the students found testing of a 100%-bug- guarantee executable more exciting and learning more effective. As a result of involving industry, every year we have a few students who choose for a graduation project at the companies who held guest-lectures.

Of course, implementing all these innovative ideas come with a price. A strong permanent technical support is needed from both the mechanical and electronics workshops. The material investment is also substantial. For example, the train setup and the water plant costed together around 20.000 euros. The first version of VU-BugZoo was financed by a grant of 50.000 euros. A more robust, extensible version will need another capital injection of around 200.000 euros.

V. Conclusions and Future Work

We presented a course in software testing read at the Vrije University in Amsterdam. We have been teaching this course for more than ten years, in an rather "unorthodox", multi-disciplinary way that combines testing with safety science, electronics, physics, healthcare and all kinds of engineering. The most effective interventions were: (1) to scare students by analyzing past, software-related accidents; (2) to thrill them using bug-hunting gamification; (3) to trust them an end-to-end testing of miniature replicas of safety-critical software-intensive systems and (4) to inspire their career, by opening a dialog with industry. The credo that drives us is that like in medicine, the more bugs a student sees, the better she knows how to fight them. In our opinion, bugs are not only ugly threats that need to be as soon as possible eradicated, but also trophies that must be understood and marveled. If we do this, one day they will become our best allies and help us to educate responsible and passionate testers. Evaluations show that this formula works. Students like the course and leave the university with a positive attitude towards testing. However, more efforts will be needed to increase their willingness to pursue a career in the field.

In the future, beside improving our own course, we want to promote teaching testing to other non-CS communities. Also, we would like to dive into the psychology of software makers, testers and users, and find answers to the question *"Why do well-intentioned people make mistakes?"*.

Acknowledgment

I would like to thank Hans van Vliet, for his pioneering idea to start a software testing course at the VU; Nancy Leveson (MIT), who inspired me in adopting a systems-centric approach. Jaap van Ekris (Delta Pi) for his lessons in risk based testing. Both VU Electronic Engineering and VU Mechanical Instrumentation groups, for crafting our wonderful systems under test. The VU CS department for financing all our crazy ideas. James Hamblen, Neil Harrison, Adithya Mathur and Tim Wilmshurst, for their help in fabricating challenging assignments. VU-BugZoo project was funded by the NRO, The Netherlands Initiative for Education Research, as part of a Comenius Teaching Fellow grant. I am very proud of, and grateful to the VU-BugZoo team, with Marc Went, Glenn Visser, Rob Limburg, Robert Jansma, Lars Eeuwijk, Viktor Bonev, Emil Apostolov, and Niels Althuisius. Finally, many thanks to all my students and teaching assistants, who contributed with stoicism, dedication and constructive suggestions to making this course better.

References

[1] M. Weiser, “The computer for the 21st century,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, p. 3–11, Jul. 1999. [Online]. Available: https://doi.org/10.1145/329124.329126

[2] T. Astigarraga, E. M. Dow, C. Lara, R. Prewitt, and M. R. Ward, “The emerging role of software testing in curricula,” in *2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments*. IEEE, 2010, pp. 1–26.

[3] H. Shah and M. J. Harrold, “Studying human and social aspects of testing in a service-based software company: Case study,” in *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2010, p. 102–108. [Online]. Available: https://doi.org/10.1145/1833310.1833327

[4] P. Waychal and L. F. Capretz, “Why a testing career is not the first choice of engineers,” *ArXiv*, vol. abs/1612.00734, 2016.

[5] A. Deak, T. Stålhane, and D. Cruzes, “Factors influencing the choice of a career in software testing among norwegian students,” *IASTED Multiconferences - Proceedings of the IASTED International Conference on Software Engineering, SE 2013*, 03 2013.

[6] A. Deak, T. Stålhane, and G. Sindre, “Challenges and strategies for motivating software testing personnel,” *Inf. Softw. Technol.*, p. 1–15, 2016. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.01.002

[7] L. F. Capretz, P. Waychal, J. Jia, D. Varona, and Y. Lizama, “Studies on the software testing profession,” in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, 2019, p. 262–263. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00105

[8] L. F. Capretz, S. Basri, M. Adili, and A. Amin, “What malaysian software students think about testing?” *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020.

[9] Y. Lizama-Mué, D. Varona, P. Waychal, and L. Capretz, *The Unpopularity of the Software Tester Role Among Software Practitioners: A Case Study*, 2020, pp. 185–197.

[10] R. Pham, S. Kiesling, L. Singer, and K. Schneider, “Onboarding inexperienced developers: struggles and perceptions regarding automated testing,” *Software Quality Journal*, vol. 25, pp. 1239–1268, 2016.

[11] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2010.

[12] A. Mathur, *Foundations of Software Testing*. Addison-Wesley, 2008.

[13] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*, 3rd ed. Hoboken and N.J: John Wiley & Sons, 2012.

[14] M. Pezzè and M. Young, *Software testing and analysis - process, principles and techniques.* Wiley, 2007.

[15] P. C. Jorgensen, *Software Testing: a Craftsman’s Approach*, 4th ed. Auerbach Publications, 2013.

[16] “Boeing 737 MAX: What’s Happened After the 2 Deadly Crashes,” https://www.nytimes.com/interactive/2019/business/boeing-737-crashes.html, accessed: 2020-10-10.

[17] N. Silvis-Cividjian, “A safety-aware, systems-based approach to teaching software testing,” in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, p. 314–319. [Online]. Available: https://doi.org/10.1145/3197091.3197096

[18] N. Silvis-Cividjian, “Teaching Internet of Things (IoT) Literacy: A Systems Engineering Approach,” in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, 2019, p. 50–61. [Online]. Available: https://doi.org/10.1109/ICSE-SEET.2019.00014

[19] “Software Testing@VU Amsterdam,” accessed: 2021-03-10. [Online]. Available: https://drive.google.com/drive/u/1/folders/1dFRuWemQ7EiKN1fHCclrPGodW0vF1oql

[20] P. Coy and M. Lewyn, “The day that every phone seemed off the hook,” *Business Week*, pp. 39–40, 1990.

[21] N. G. Leveson and C. S. Turner, “An investigation of the Therac-25 accidents,” *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[22] N. G. Leveson, *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.

[23] N. Silvis-Cividjian, R. Limburg, N. Althuisius, E. Apostolov, V. Bonev, R. Jansma, G. Visser, and M. Went, “VU-BugZoo: A Persuasive Platform for Teaching Software Testing,” in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE ’20, 2020, p. 553. [Online]. Available: https://doi.org/10.1145/3341525.3393975

[24] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, “Hints on test data selection: Help for the practicing programmer,” *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[25] S. Elbaum, S. Person, J. Dokulil, and M. Jorde, “Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable,” in *29th International Conference on Software Engineering (ICSE’07)*, 2007, pp. 688–697. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE.2007.23

VI. Author’s profile

Natalia Silvis-Cividjian is lecturer in the Computer Science department at the Vrije Universiteit (VU) in Amsterdam, The Netherlands. She is a computer engineer, with a PhD degree in applied physics. She has been teaching for more than 10 years Pervasive computing for 1CS and Software Testing for graduate SE students. In fact these two courses complement each other - one is teaching how to specify, design and create smart systems and the other one how to make them robust and safe. Both courses are taught in a multidisciplinary approach, combining core CS topics such as programming, testing and machine learning, with engineering topics such as signal processing, control theory, physics and electronics, wrapped together in an unifying safety science shell. Both courses have a strong practical component, where students work with physical models of real-life, smart and safety-critical systems, such as autonomous vehicles, trains and medical devices. She published a few papers on innovation in CS education at ITiCSE and ICSE conferences. Fascinated by accidents in safety-critical systems, such as the Therac-25 radiotherapy machine mishaps, she applied a new systems theory based safety analysis approach called STAMP to this particular field. The results were published in a paper in Safety Science. She also supervised multiple MSc projects in the software testing field, with companies such as Oce, KLM, ING, Sogeti and Sioux. In 2019 she won a Comenius Teaching Fellow grant from the Dutch Ministry of Education to develop VU-BugZoo, a learning platform to improve software testing teaching. In 2017 she published a textbook with Springer with the title ”Pervasive Computing: Engineering Smart Systems”.