2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG) | 979-8-3503-0186-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/SEENG59157.2023.00007

Hey Teachers, Teach Those Kids Some Software Testing

Baris Ardic
Delft University of Technology
Delft, The Netherlands
b.ardic@tudelft.nl

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
a.e.zaidman@tudelft.nl

***Abstract*—Software testing is generally acknowledged to be an important weapon in the arsenal of software engineers to produce correct and reliable software systems. However, given the importance of the topic, little is known about where software engineers get their testing knowledge and skills from. Is this through (higher) education, training programmes in the industry, or rather is it self-taught? In this paper, we investigate the curricula of 100 highly ranked universities and survey 51 software engineers to shed light on the state-of-the-practice in software testing education, in terms of both academic education and education of software engineers in the industry.**

***Index Terms*—software testing, software engineering, education**

I. Introduction

Software testing is essential to ensure the quality of the software systems that we as a society rely on [1]. Inadequate testing can have severe consequences, with repercussions of software failures ranging from users getting frustrated [2], over huge financial losses [3] or years of lost research [4], to causing injury or even death [2]. As Aniche et al. state "Making sure software works is maybe the greatest responsibility of a software developer" [5].

With this great responsibility for software engineers, we wonder where and how software engineers acquire their software testing knowledge and skills. In particular, we are curious to better grasp what the role of higher education is in teaching budding software engineers about software testing. But we are equally interested in understanding how the work environment, or professional education, contributes to the software testing skills of practitioners. As such, our curiosity-driven investigation approaches this from two angles:

1) We investigate the curricula of the top 100 universities in the "computer science subject ranking" of the Times Higher Education [6]. In this investigation, we look through their curricula to find dedicated testing courses.
2) We survey 51 practitioners to better understand where they acquired their testing knowledge and skills from.

Our investigation is steered by these research questions:

RQ1 How common are dedicated software testing courses in the curricula of the universities that we consider? How common is it for practitioners to have followed a dedicated testing course?

RQ2 What are the topics being discussed in these dedicated testing courses? What are testing topics that are in demand from practitioners?

RQ3 What are the learning objectives of these dedicated testing courses?

RQ4 How and where do practitioners acquire their testing knowledge and skills from?

Our high-level observations are that around half of the 100 computer science curricula that we have screened contain a dedicated testing course. We also observe that there is a high demand among practitioners to get more education on testing subjects and that the most popular way for practitioners to learn about new software testing techniques is to learn it by themselves or learn it from peers.

This research is sponsored by the Swiss National Science Foundation (SNSF Grant 200021M_205146) and the Vici "TestShift" project (No. VI.C.182.032)

II. Related Work

The academic community is well aware of the gap between software testing education and the industry's expectations. We structure the earlier research around two axes, namely: (1) higher education curricula, and (2) industry experience.

A. Software Testing Curricula Related

Garousi et al. have performed a systematic literature mapping study on papers related to software testing education [7]. Their analysis provides a broad overview of educational approaches in software testing education. They have also summarized the contributions in terms of pedagogical approaches for software testing education.

Zhu and Zhang propose structural reforms for a software testing course where they divide it into instruction and practical stages, involving case-based teaching and project-driven employment-oriented training [8].

Clarke et al. augment a testing course with testing tool tutorials and observe improvements in student test suites when code coverage tools are used [9].

Deng et al. designed five auxiliary resources to be used in testing courses. These resources use open-source software to improve the hands-on testing experience the course provides. These additional activities span multiple aspects of software testing from unit testing to test driven development [10].

979-8-3503-0186-1/23/$31.00 ©2023 IEEE
DOI 10.1109/SEENG59157.2023.00007

Aniche et al. investigate the prevalent challenges, mistakes, and favourite activities of their students in their testing course. Subsequently, they pragmatically reengineer their course to fit the goals obtained from the student experiences [5].

B. Analysing Industry Expectations

Garousi et al. surveyed practitioners to establish testing challenges in the industry. Their findings point to test management and automation being the more challenging aspects of software testing [11].

Florea and Stray directly look into what employers demand from software testing-related positions. To do so, they mine job advertisements and analyse them. Their analysis shows test design, execution, and planning to be the most consistently sought-after skills [12].

Hynninen et al. designed an outline for a testing course using a constructive alignment approach. In order to determine their learning objectives, they constructed a survey regarding testing processes in the Finnish software industry. This information was then used to create industry-facing learning objectives and teaching methods [13].

III. Study Set-Up for the Curriculum Analysis

A. Dedicated Software Engineering Courses

We assume that for many budding software engineers, higher education is the first time that they come into contact with software testing. This might happen as part of their introductory programming courses, or in a course dedicated to software quality assurance (or even software testing entirely). In this study, we focus on courses of which at least 50% of its contents are about software testing, the so-called *dedicated software testing courses*. We define the ones that teach some software testing, but are below this threshold, as partial testing courses. We focus on dedicated courses because we believe that the combination of the importance of software testing, and the amount of important topics to be discussed in it, requires a course that dedicates sufficient time to it.

B. Course Selection

We start our investigation by looking into the top 100 universities in the *computer science and engineering subject rankings* by Times Higher Education (THE) [6]. We manually searched for software testing courses in the catalogues of these universities (we coined this the THE100 dataset). In some instances, the course catalogue was unreachable, incomplete or outdated. For these cases, we used auxiliary search strings that directly target dedicated testing courses.

Through this process, we found 49 dedicated and 34 partial testing courses. For 18 universities, we have not been able to detect any testing-related courses. This is either caused by an actual lack of courses or a lack of publicly available information on the offerings of the institution.

These dedicated testing courses include both graduate and undergraduate level courses. We do not distinguish between them because this information is frequently not available or hard to infer. There are regional differences that significantly change the rate of obtaining a graduate degree which decreases the value of making a distinction.

C. Data Collection

To further investigate the contents of dedicated testing courses, we attempted to find their syllabi. We found a syllabus for each course, except for one, but the level of detail that we could find per course varied considerably. The documents obtained for each course range from a complete and detailed syllabus to a short description. In the rest of this study, all of these findings are referred to as course syllabi. We utilized manual keyword extraction while separating the outcomes from course descriptions. Some syllabi also listed their own keywords in the form of lecture-topic schedules or talking points. When they were available, we preserved those keywords. All methods, concepts, and tools that relate to software engineering are extracted as keywords. There are 13 description and 2 outcome keywords on average per syllabus. Study data is available in the replication package [14].

IV. Results of the Curriculum Analysis

A. Topics Taught and Co-Occurring Topics

During our investigation of dedicated testing course syllabi, we observed the topics that were being taught alongside software testing. These topics are mainly verification, code review, risk management, and DevOps. There are also some courses that include recent research topics or outputs from the software testing academic community (e.g., fuzzing). Understanding which topics are frequently taught next to software testing enables us to better understand modern software testing education. Figure 1 depicts an up-scaled set representation of the topics taught. The figure demonstrates the intersection sizes in and between each category. Additionally, if a course does include any additional topic they are included in the first intersection category. Some interesting observations are:

- There are 9 dedicated testing courses that include software engineering research in their syllabus. Teaching software engineering research-oriented material is as common as risk management and DevOps combined. This indicates that the popularity of software testing as an academic field is also reflected in the syllabi. Almost all of these courses are graduate-level and have some research-oriented outcomes.
- There are 16 courses that focus only on software testing, while 19 courses combine software verification and testing. Teaching software testing and verification together is more common than teaching just software testing.

B. Regional Course Availability

In the overall THE100 dataset, we notice that around 50% of the universities have a dedicated testing course. However, we observe a regional disparity: ∼61% of the European universities have a dedicated testing course, ∼49% of the North American universities, and ∼25% of the Asian universities. The low percentage in Asian universities might be due to the difficulty in accessing the information on university websites

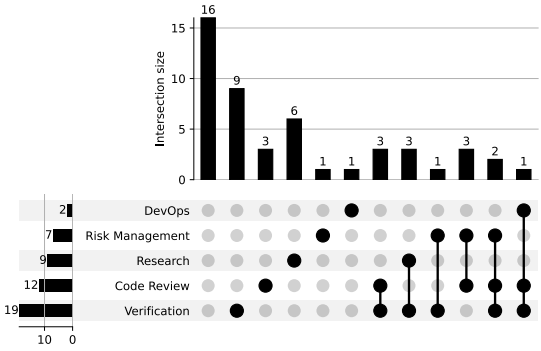

Fig. 1. Distribution of topics taught alongside testing

in Asia, even though we asked a colleague for help with the language barrier.

C. Common Resources

We analysed the course descriptions to find out which recommended resources (including textbooks) are listed. We found that the 4 most popular resources are textbooks (starting with their number of occurrence):

5: *The Art of Software Testing* by Myers et al. [15].
5: *Introduction to Software Testing* by Ammann and Offutt [16].
4: *Software Testing and Analysis* by Pezzè and Young [17].
3: *Software Testing* by Patton [18].

There are about 120 unique resources that were cited in the collection including books, research papers and grey literature. Other resources appeared in at most two courses. The full list is available in our replication package [14].

D. Common Outcomes

In terms of course outcomes or learning objectives, 32 out of the 49 syllabi of dedicated testing courses in our collection report them. Most of the text related to outcomes consists of abstract and verbose sentences. This is quite different from the almost list-like qualities of course descriptions. Therefore our evaluation of the common outcomes is done slightly differently than the rest of the syllabus evaluations.

There are various generic learning outcomes that are put into course syllabi such as "communicate effectively in English for general project presentation" [14, case 33]. We ignore these and evaluate software engineering outcomes by clustering the individual outcomes manually. The most common outcome categories are demonstrated below with examples and the number of occurrences:

- **Practical experience (25)** - Participants are able to write maintainable test code by avoiding well-known test code smells [14, case 30].
- **Testing techniques (17)** - Understand the range of approaches to testing that can be applied to software systems [14, case 13].
- **Tools and frameworks (12)** - Getting to know methods and tools of software testing [14, case 31].
- **Fundamentals of software testing (11)** - Know the goals, concepts, models, and basic terms of software quality assurance [14, case 28].
- **Quality assurance techniques (11)** - Understand the strengths and weaknesses of different quality assurance techniques [14, case 3].
- **Proper technique (10)** - Evaluate the suitability of different techniques for a given software and set of constraints [14, case 18].
- **Testing strategy and plan (9)** - Design and implement strategies for testing software in structured and organized ways. [14, case 43].
- **Measure quality (8)** - Identify appropriate quality goals for moderately sized software [14, case 46].
- **Research (8)** - Critically evaluate research papers in the field of program testing and analysis [14, case 26].
- **Automated testing (7)** - Understand how automated testing and analysis techniques work [14, case 20].

The above list indicates that practical experience with testing, testing techniques, and tools are highly valued by educators. Some other interesting observations are that understanding and utilizing quality assurance techniques, which include testing and static quality analysis, is one of the top categories. This corresponds to the observation made in Figure 1 where we see that code review is a popular topic.

Another prevalent learning outcome is the selection of the proper quality assurance technique, likely instigated by the plethora of software testing techniques that are available.

E. Software Testing Skills

We investigate software testing skills in multiple categories. The overall structure of these categories is intended to match an existing structure from Florea and Stray [12]. This structure is built from software testing job advertisements and the ISTQB Foundation Level syllabus [19]. Figure 2 provides an overview of the distribution for these software testing skill categories. We extract keywords from each syllabus and further organise these keywords into the following five categories:

- **Static Testing** covers all forms of code inspection and other static quality testing methods.
- **Test Level** covers different scopes of testing which are unit, integration, system and acceptance testing.
- **Test Process** consists of keywords related to automated testing, test design, test plan, and defect management.
- **Test Tools** include keywords that relate to all test-related tools, frameworks, and libraries.
- **Test Type** has all the different techniques and methods taught in our collection of courses.

Our analysis of the syllabi aims to avoid evaluating data at a keyword level. This is because a syllabus is a loosely defined type of document with varying levels of information. Therefore, a syllabus failing to mention a specific topic does not necessarily mean that it is not addressed.

V. Study Set-up of the Practitioner Survey

In order to get a glimpse of what practitioners think of software testing education at the academic level, but equally

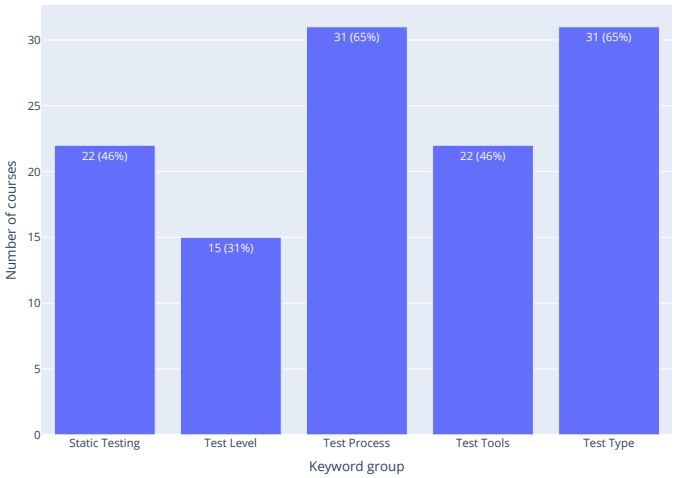

Fig. 2. Distribution of software testing skills

education that happens within a professional environment, we have conducted a short online survey. The intended participants for this part of the study are people who regularly carry out software testing-related activities. The survey is distributed via convenience sampling where the most frequently utilized channels are social media platforms and online message boards, e.g., Twitter, Reddit, and LinkedIn. This sampling method is common among empirical software engineering studies [20] and is also appropriate for this study because the focus is on exploration instead of assuring generalizability. The responses were collected between December 1st and 31st of 2022. Some potential participants only responded to the demographics questions, after omitting these we were left with 51 actual responses. Since none of the questions are required to be able to submit the survey, throughout the analysis the number of respondents per question might differ slightly. Conversely, people with a high school degree only did not get specific questions on courses during university.

Our survey broadly explores the participants' experience with software testing education, both at the academic level and on the job. We gauge what testing activities they frequently perform, which ones they find difficult, and which ones they would like to have learned more about. A summary of the questions that we have asked in our survey can be found in Table I; throughout the rest of this paper, we refer to this table by referencing the question numbers in parentheses.

TABLE I
SUMMARY OF PRACTITIONER SURVEY QUESTIONS

Q1	Consent
Q2–8	Demographic questions
Q9	How frequently do you perform the following types of software testing activities?
Q10	In your opinion, which testing activities are more difficult to adequately perform?
Q11	How frequently have your projects been negatively impacted by your or your colleagues' lack of software testing capabilities?
Q12	Have you ever taken a university course that dedicated any time to teaching software testing?
Q13	How much time did this course spend on testing?
Q14	Which testing activities would you have liked to learn more about at your university?
Q15	Please rank the methods below from most impactful to least impactful regarding your software testing education. If you are unfamiliar with a source rank it last.
Q16	How often do you read technical papers or articles on new developments in software testing?
Q17	Select any resources that you use regularly for testing knowledge-related issues.
Q18	Please specify the resources that you had in mind in the previous question.
Q19	"Software testing should not be taught during higher education programs." Do you agree with this statement?
Q20	Why did you agree with the statement?
Q21	How was your first experience applying testing practices in a professional context? What had the biggest impact on how you experienced testing practices at that time compared to today?

VI. RESULTS OF THE PRACTITIONER SURVEY

A. Participant Demographics

The participants of the survey were asked several questions to determine their experience and background. In this section, we present the data obtained from these questions.

- **Current position:** The participants could pick any number of fitting roles from a predefined list; they could also opt to fill in a role not on our list. The most selected options were software developer (20), quality assurance (11), software tester (9), and higher management (7). All other options were selected less than twice.
- **Subject experience:** Participants indicated that they have on average 7.4 years of software development experience. Software testing experience is at 7.7 years on average, mainly because Q&A profiles have more experience.
- **Education background:** The most frequently obtained degrees obtained by the respondents are: Bachelor's (21), Master's (18), High school (5) and Doctoral (3). The most common subjects of study are computer science (16), software engineering (4), informatics (2), and software development (2). These degrees are obtained in institutions established in 20 countries with the most popular options being the Netherlands (8), the United States (6), Turkey (6), and the United Kingdom (4).
- **Company Size :** The most common type of workplace had more than 500 employees (21), followed by 51-500 (13), and less than 50 (10).

B. Software Testing Activities

This section of our survey aims to investigate participants' experiences with common testing activities. To do so, some of the upcoming questions utilize a list of software testing activities that we compile together from "test type" and "test level" in the taxonomy provided by Florea and Stray [12]. We have extended the nonfunctional testing activities to be more specific since we have not encountered this analysis before. The final list consists of unit, integration, system and acceptance testing coming from "test level" and functional,

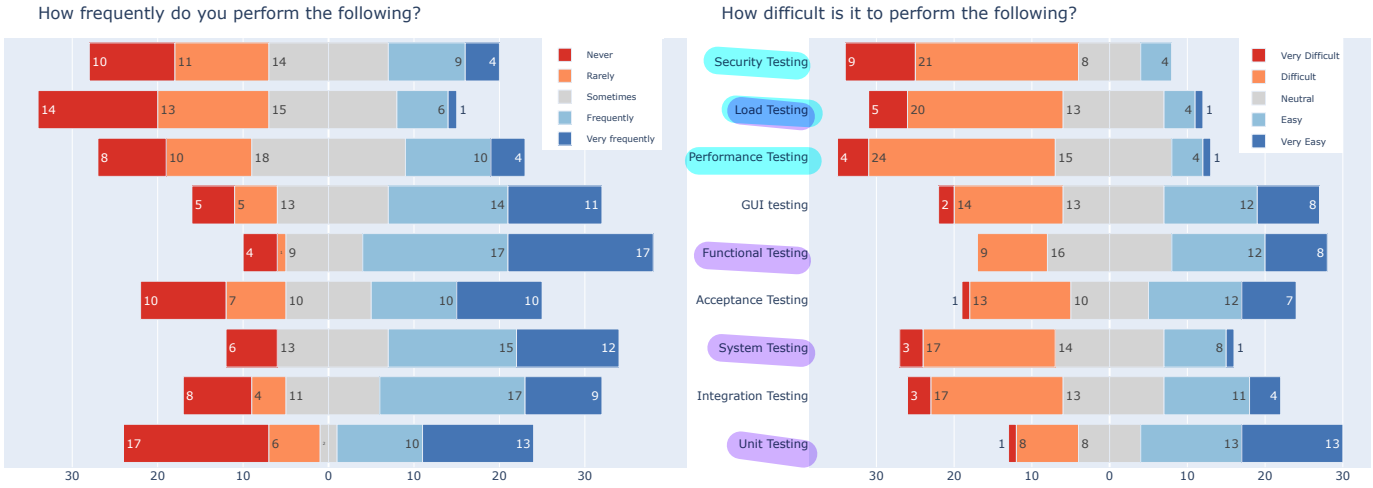

Fig. 3. Frequency and difficulty of software testing activities

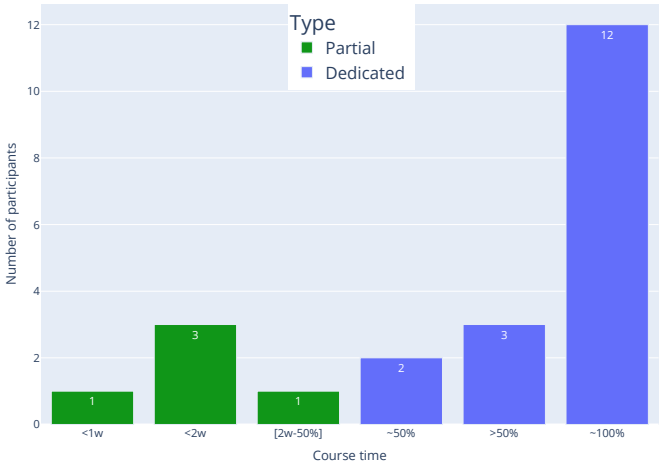

Fig. 4. Time spent on testing from the courses taken by participants

security, load, performance and GUI testing from "test type". The first question was about the frequency of these activities [**Q9**], which was followed by their relative difficulty [**Q10**].

When we investigate the results presented in Figure 3, we identify security, load, and unit testing as the least frequently performed activities. Moreover, functional and system testing are the more frequently carried out testing activities. When we turn our attention to the right side of Figure 3, we observe that security, load, and performance testing are considered to be more difficult testing activities; functional and unit testing are considered as easier activities.

When we focus on unit testing, we observe in the left side of Figure 3 that the number of respondents that indicate to perform unit testing is balanced, i.e., around half do unit testing frequently, while the other half do not. When we cross-check this with the profiles of the respondents, we observe that those respondents that do not perform unit testing (frequently) are mostly QA-engineers, while the group that (frequently) unit tests are mostly software engineers.

The next question asks about respondents' frustrations: "How frequently have your projects been negatively impacted by your or your colleagues' lack of software testing capabilities?" [**Q11**] Figure 5 shows the distribution of answers: ∼40% of participants are (very) often negatively affected. Only one participant reports that their projects are never affected. The overall results indicate that almost all participants have observed or experienced shortcomings in testing competency.

C. Participants and Testing Courses

In section III-A we have established the concept of a dedicated testing course. During our analysis of the curricula of the top-ranked universities, we have tried to establish the presence of such courses in the curricula. Now, we augment our findings by asking the respondents whether they participated in such a course.

First, we asked the participants whether they have taken any course that dedicated any time to teaching software testing [**Q12**]. Around 45% (22) of the participants had taken such a course, while ∼49% (24) did not. Three participants did not remember whether they had taken a software testing course or not. Upon closer inspection, we found out that participants who took testing courses were mostly from computer science

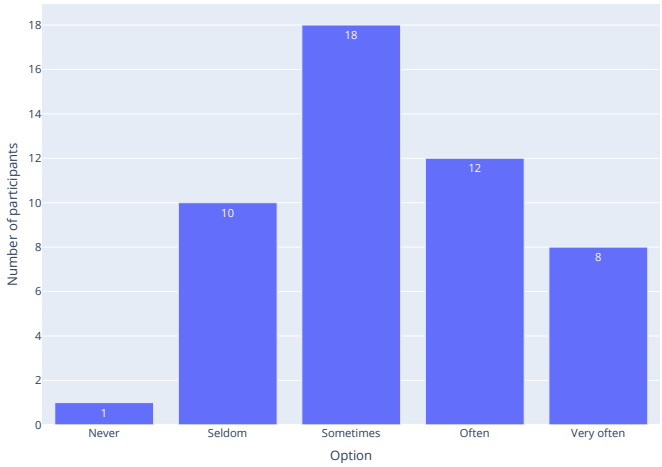

Fig. 5. Negative impact of lack of capabilities on projects

(11) and software engineering (4) majors. Moreover, when we cross-examine these results with overall degree types from the participant demographics, we see that ~69% of the computer science graduates and all of the software engineering graduates had taken a testing-related course.

We asked these participants to classify the course they took in terms of the amount of time spent on learning testing [**Q13**]. The question had options ranging from one or two weeks spent on testing, to half or more of the total time. Figure 4 lays out the information we collected. Most of the courses taken by the participants are dedicated testing courses.

The next question directed to the participants was "Which testing activities would you have liked to learn more about at your university?" [**Q14**] This question was directed to all participants, not just the ones who had taken dedicated testing courses. Figure 6 visualises the participants' preferences.

Globally, we can observe that the "I would have liked to learn more" is frequently chosen across all types of testing that we presented to the respondents. The top-3 of testing types that participants want to learn more about are respectively security, load, and performance testing.

The activities that are most adequately known by the participants are integration, functional and unit testing indicating that these topics are handled well by higher education.

To provoke the respondents to our survey, we also asked them if they agree with the following statement: "Software Testing should not be taught during higher education programmes" [**Q19**]. Around 92% (45) of the participants said "No, it should be taught", reinforcing our intuition of the necessity of testing education.

D. Practitioner Sources

The rest of the survey included questions about the learning resources of the practitioners. To discover the most impactful learning resources of the participants, we gave them the following list and asked them to rank it from most impactful to least impactful [**Q15**].

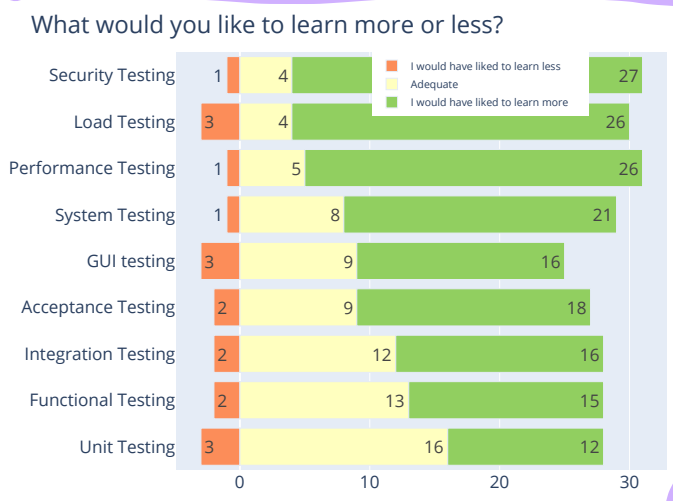

Fig. 6. The learning preferences of participants per testing activity

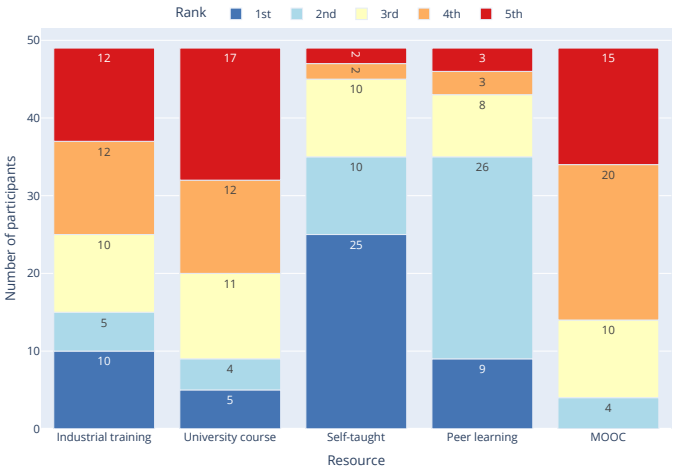

Fig. 7. Most impactful learning resources of the participants

- Industrial training
- University course
- Self-taught
- Peer learning (learning from colleagues, etc.)
- Massive open online courses (MOOC e.g Udemy, Coursera)

Figure 7 demonstrates our findings as a stacked bar chart where each bar represents a learning resource. For each bar, starting from the bottom with the first rank, we can see how many participants placed a resource in a particular position.

For example, we can see that "self-taught" category is the most popular option with about half of the participants ranking it as their primary source of testing knowledge and skill. The most popular second-ranked item is "peer learning". The rest of the ranks do not have a dominant option. However, we can still observe that "MOOC" is almost never picked for the first two ranks. "Industrial training" and "University course" have very similar distributions.

In a following question, we asked them how frequently they read technical papers or articles related to software test-

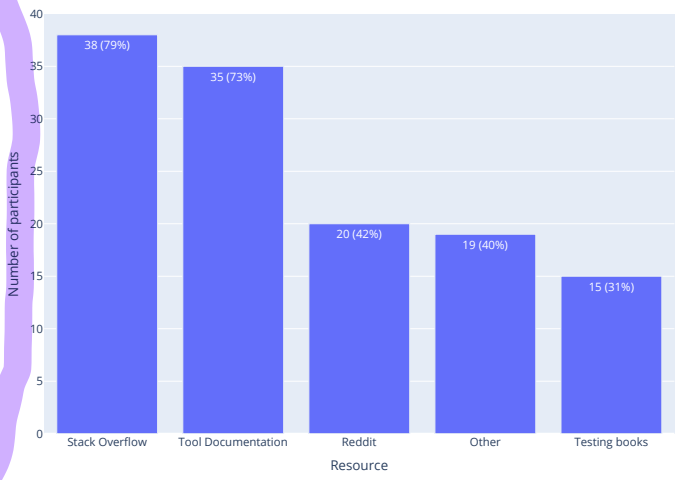

Fig. 8. Regularly used resources

ing [**Q16**]. According to our results, ∼43% of the participants read at least one related article per month, while ∼23% never read any related literature.

Moreover, we asked participants which testing-related resources they regularly use for testing knowledge-related issues [**Q17**]. The predefined resources and their popularity is shown in Figure 8. Keep in mind that participants were able to select all of the resources they were using. We allowed them to add their own resources as well. The most popular option that we did not include in our list was "software testing related blog" which was mentioned by six participants.

VII. Discussion & Conclusion

A. Revisiting the Research Questions

RQ1: *How common are dedicated software testing courses in the curricula of the universities that we consider? How common is it for practitioners to have followed a dedicated testing course?* In our initial sample of 100 universities, we found 49 dedicated testing courses. However, the existence of one does not mean that it is always offered; we did not have a feasible way of looking into the aspect of course availability. Comparing this to the results of the survey, 17 (∼35%) participants had taken a dedicated software testing course (see Figure 4). The decrease in the percentage can be explained by variables like student interest or course availability.

RQ2: *What are the topics being discussed in these dedicated testing courses? What are testing topics that are in demand from practitioners?* The most popular testing skill categories are "test process" and "test type" (Figure 2). These categories of keywords can be observed in ∼65% of the dedicated testing courses. Previous studies on testing education had already drawn attention to automated testing, test design and planning [11], [12], which are included in the "test process" category. In this study, we observed that security, load, and performance testing are in high demand as they are the top testing types that participants wanted to learn more about (Figure 6) at university. These are also seen as harder to perform by the participants in Figure 3. These findings together point to a need for further investigation regarding budding engineers' opportunities regarding learning these skills.

RQ3: *What are the learning objectives of these dedicated testing courses?* In the syllabus analysis, we observed that the common course outcomes were about practical experience, techniques, tools and fundamentals. These objectives create the backbone of the testing knowledge that these courses aim to convey. However, having a specific outcome such as "proper technique" high on the list points to the sheer number of techniques that are available.

RQ4: *How and where do practitioners acquire their testing knowledge and skills from?* In our survey with 51 software testing practitioners we have observed that individual learning approaches like "self-taught" and "peer learning" are more popular than more organized ways of learning like industrial training or university courses; this also includes MOOCs. Popular resources for software testing practitioners to learn more are StackOverflow and tool documentation. An important prerequisite question when studying knowledge gathering from practitioners is then obviously whether there is a need for additional learning. In Figure 6 we can see that most participants think they could have learned more about many testing activities.

B. Implications

The general implications of our study are that an important computer science topic like software testing is taught through a dedicated course in about half of the 100 computer science curricula that we analysed. We acknowledge that this is not the only way to teach software testing, e.g., through project work, but it does *raise the question of whether the wide range of software testing techniques is conveyed to students*.

When we look at the results from the survey among practitioners, we see a general need for more knowledge in many areas of testing ranging from systems and acceptance testing to security, load, and performance testing. The latter three categories of testing are interesting because they obtain the most votes in terms of testing activities that practitioners want to learn more about (practitioners also consider these to be more difficult). We see two potential reasons for this: (1) it could be that these topics have gained more importance recently, for example, because load testing is more important for software that is offered as a service, or because of the additional security requirements that need to be tested for preventing cybersecurity threats, and (2) it could be that higher education does not prepare students for these types of testing.

In terms of how software testing practitioners gain new knowledge and skills, it is very apparent that the respondents to our survey clearly *prefer to learn new software testing techniques either by themselves or from peers*. More organised forms of education are not appreciated as much. This potentially calls for a rethink in how we offer software testing education to practitioners.

money lines

C. Threats to Validity

Our study is exploratory in nature, and therefore we acknowledge that there are threats to the validity of our conclusions. We will now discuss the most important ones:

External validity. For the syllabus collection process we have used the Times Higher Education (THE) subject-based rankings for Computer Science [21]. We opted to use a higher education institution ranking to circumvent having to use random sampling. We are aware that taking highly ranked institutions might skew the data towards having more dedicated testing courses; we consider this an interesting avenue for further investigation.

For our survey, we used convenience sampling to recruit participants. We are aware that this recruitment method might skew our results, as it is likely that only people with a clear interest in software testing have filled out the survey. We have potentially observed this phenomenon in our results, as the majority of the testing courses taken by the respondents

were dedicated testing courses, which might indeed indicate an elevated interest in software testing from the participants. In future research, we should set out a broader recruitment campaign to avoid sampling bias.

Construct validity To ensure the validity of the syllabus analysis process, the keyword extraction process was done in three passes. We identified course descriptions, outcomes, and resources in each syllabus document in the first pass and recorded them. In the later passes, we performed keyword extractions for descriptions and outcomes. We merged the keyword groups that we identified into larger groups ending up with the data that we used for analysis. Throughout the process, the authors checked each other's work.

We can not argue for the completeness of the syllabi: if a syllabus document does not mention a particular topic, it does not mean that the topic is not addressed. Vice versa, if a resource is mentioned in the syllabus, it does not mean that it is extensively used in the course.

D. Future Work

The implications of this study show a need for further investigation into the learning process of non-functional software testing. The subjects of security, performance and load testing require additional domain knowledge to comprehend and teach. Creating resources to aid in teaching these subjects effectively is, therefore, an undertaking that is required from the software engineering community.

We also intend to further explore the process of learning software testing by utilizing participants' perspectives starting with a qualitative analysis of the experiences reported in [**Q21**] which was out of the scope of this study. We aim to conduct interviews with key individuals on the details of the process of learning software testing.

REFERENCES

[1] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann, and A. Zaidman, "Developer testing in the IDE: patterns, beliefs, and behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 3, 2019.

[2] A. J. Ko, B. Dosono, and N. Duriseti, "Thirty years of software problems in the news," in *Int'l Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM, 2014, pp. 32–39.

[3] S. Matteson, "Report: Software failure caused 1.7 trillion in financial losses in 2017," 2018. [Online]. Available: https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-inancial-losses-in-2017/

[4] G. Miller, "A scientist's nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, 2006.

[5] M. Aniche, F. Hermans, and A. van Deursen, "Pragmatic software testing education," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 414–420.

[6] Times Higher Education. Subject rankings - computer science. [Online]. Available: https://www.timeshighereducation.com/world-university-rankings/2022/subject-ranking/computer-science

[7] V. Garousi, A. Rainer, P. Lauvås Jr, and A. Arcuri, "Software-testing education: A systematic literature mapping," *Journal of Systems and Software*, vol. 165, p. 110570, 2020.

[8] B. Zhu and S. Zhang, "Curriculum reform and practice of software testing," in *Int'l Conf. on Education Technology and Information System (ICETIS)*. Atlantis Press, 2013, pp. 846–849.

[9] P. J. Clarke, D. L. Davis, R. Chang-Lau, and T. M. King, "Impact of using tools in an undergraduate software testing course supported by WReSTT," *ACM Transactions on Computing Education (TOCE)*, vol. 17, no. 4, pp. 1–28, 2017.

[10] L. Deng, J. Dehlinger, and S. Chakraborty, "Teaching software testing with free and open source software," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 412–418.

[11] V. Garousi, M. Felderer, M. Kuhrmann, and K. Herkiloğlu, "What industry wants from academia in software testing? hearing practitioners' opinions," in *Proc. of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2017, pp. 65–69.

[12] R. Florea and V. Stray, "The skills that employers look for in software testers," *Software Quality Journal*, vol. 27, no. 4, pp. 1449–1479, 2019.

[13] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Guidelines for software testing education objectives from industry practices with a constructive alignment approach," in *Innovation and Technology in Computer Science Education*. ACM, 2018, pp. 278–283.

[14] B. Ardic and A. Zaidman, "Replication package for "Hey teachers, teach those kids some software testing"," 2023. [Online]. Available: https://doi.org/10.6084/m9.figshare.21895701.v3

[15] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[16] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge University Press, 2016.

[17] M. Pezze and M. Young, *Software Testing and Analysis: Process, Principles and Techniques*. Wiley, 2008.

[18] R. Patton, *Software testing*. Pearson Education India, 2006.

[19] F. ISTQB, "Foundation level syllabus version 2011," *International Software Testing Qualifications Board*, 2011.

[20] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, pp. 1–31, 2022.

[21] Times Higher Education. University rankings. [Online]. Available: https://www.timeshighereducation.com/world-university-rankings

Baris Ardic received the BSC and MSc degrees in Computer Engineering from Bilkent University, Turkey. He is currently a PhD candidate in the Software Engineering Research Group (SERG) at the Delft University of Technology. His research interests include software engineering education, software testing and empirical software engineering.

Andy Zaidman received the MSc and PhD degrees in computer science from the University of Antwerp, Belgium, in 2002 and 2006, respectively. He is currently a full professor with the Delft University of Technology, The Netherlands. In 2013, he was the laureate of a prestigious Vidi mid-career Grant and in 2019, the most prestigious Vici career grant from the Dutch science foundation NWO.