Testing Education: A Survey on a Global Scale

Silvana M. Melo, Veronica X. S. Moreira
Faculty of Exact Sciences and Technology
Federal University of Grande Dourados
Dourados, MS, Brazil
{silvanamelo,veronicaxixa}@ufgd.edu.br

Leo Natan Paschoal, Simone R. S. Souza
Institute of Mathematics and Computer Science
University of São Paulo,
São Carlos, SP, Brazil
paschoalln@usp.br,srocio@icmc.usp.br

ABSTRACT

[Background]: The software industry has a high demand for professionals in the software quality area. However, computing students, in general, leave the university with little or no knowledge in software testing. **[Aim]** Although previous studies have investigated how software testing is covered in universities, they focus on pedagogical approaches, trends, and technologies for improving the educational process, and do not evaluate in-depth the contents explored. This article addresses a worldwide perspective on the way educators both deal with the software testing subject, materials and details covered, support mechanisms and teaching practices applied, and challenges imposed, and evaluate instruments. **[Method]** A survey with questions was conducted with software testing professors from April to June 2020 towards investigating the way the subject is taught in the classroom. **[Results]** Most higher education institutions address the fundamental software testing techniques and criteria, with more completeness in specific than general software engineering courses. Additional findings and discussions are presented in detail throughout the paper. **[Conclusion]** This study provides a comprehensive overview of the software testing education field from educators' perspectives. We strongly believe it is useful for the understanding of the actual scenario of the topic, provides foundations and directions to new research, guides educators into subjects' conduction and identification of potential weaknesses and solutions to recurrent problems.

CCS CONCEPTS

• **Software and its engineering** → Software verification and validation; • **Social and professional topics** → Software engineering education;

KEYWORDS

Software Testing, Survey, Computing Education and Training

ACM Reference Format:
Silvana M. Melo, Veronica X. S. Moreira and Leo Natan Paschoal, Simone R. S. Souza. 2020. Testing Education: A Survey on a Global Scale. In *34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3422392.3422483

SBES '20, October 21–23, 2020, Natal, Brazil

ACM ISBN 978-1-4503-8753-8/20/09…$15.00
https://doi.org/10.1145/3422392.3422483

1 INTRODUCTION

Different activities in the software development process can contribute to the software quality. Among these activities, software testing aims to detect defects and analyze if the software runs as expected [11]. If defects are not identified and removed, they can cause failures, vulnerabilities, and damage to an organization and its users [20] and, in a critical system, lead to irreversible consequences (e.g., death of humans) [22]. In this perspective, the low quality of the software testing activity can exert adverse effects not previously envisaged by the software development team.

Software testing has been increasingly important in software development companies [8]; however, such organizations still find difficulties in selecting good testers [1], due to a lack of qualified human resources for tester jobs. The problem is professionals without the necessary skills cannot correctly apply the software testing activity, and fail to select good test cases, deciding on when the software is sufficiently tested and on the most critical parts to be best tested [4].

Some studies have indicated the importance of universities preparing students better for the tester career; in general, such institutions lack disciplines that include the subject of software testing in-depth [2, 3, 10]. Therefore, software engineering and software testing educators have adjusted the curriculum of computer science courses towards improving students' abilities in software testing.

Although some courses include a software testing discipline, most universities still teach software testing as a topic in a software engineering discipline [15, 21]. Some courses offer a specific software testing discipline, but often as an elective one.

Software testing is usually given little attention at universities. The testing research community has investigated its teaching [5, 21], which is fundamental for the characterization of the software testing teaching-learning, however, studies have not included the teachers' point of view.

The literature reports many discussions on the use of new pedagogical approaches for more effective learning and improvements in students' interest in software testing subjects [13, 14, 16, 19].

The understanding of the way software testing has been taught, from an educators' point of view, is fundamental. Paschoal et al. [15], identified the topics taught on software testing, the educational resources and methodologies used, and other important issues, all from a teachers' point of view. However, such an initiative was undertaken in a national context, failing to consider the opinion of educators from other countries and continents.

Secondary studies have mapped pedagogical activities, course designs and more efficient technologies in the teaching process; however they have not included all educators that, despite working in the area, do not publish on software testing education [6].

Our survey aims to complement the community's understanding of software testing teaching from the point of view of an educator who is on the front line of software testing teaching. The main contributions of this research include (i) description of the contents taught in courses which cover software testing topic (e.g., testing techniques, criteria, levels, concepts), (ii) characterization of teachers' knowledge, experience level, and location, (iii) a description of the teaching approaches used and the support mechanisms applied by educators, (iv) a list of assessment mechanisms that measure learning, and (v) the main challenges that affect the software testing teaching.

This paper addresses research planned and conducted towards a global overview of software testing teaching, from the point of view of software engineering and software testing educators from different universities and countries, and is structured as follows: Section 2 focuses on research on software testing teaching related to this study; Section 3 describes the method applied; Section 4 reports the results from the survey, according to each research question proposed; Section 5 discusses the data collected; Section 6 describes threats to validity; finally, Section 7 provides the conclusions and suggests some future work.

2 PREVIOUS STUDIES

The literature reports studies that provide an overview of software testing teaching; according to the research strategy used, they can be grouped into: (i) studies that analyze reference curriculum and curricular structures of university subjects delimited by the researchers; (ii) studies based on a survey conducted with computing graduate students that perform testing activities in software industries; and (iii) studies based on a survey conducted with software testing professors. Their main contributions and differences in comparison to our study are reported in this section.

Garousi and Mathur [5] presented an overview of software testing teaching at Canadian and American universities, analyzing the testing tools applied in computing courses offered by institutions selected by the authors, as well as software programs and systems used in practical test classes. The authors also compared the coverage of three textbooks that might be used by teachers of the test subjects offered by such institutions. They observed (i) an effective teaching of the content provides students with opportunities to solve real-world problems, which requires realistic practical exercises and most popular testing tools on the market, and (ii) teachers should select textbooks that best match the disciplines objectives.

Valle et al. [21] analyzed the reference curriculum proposed by the *Association for Computing Machinery* (ACM) and the *Brazilian Computer Society* (SBC). Search analysis aims to offer the community an understanding on the way future testing practitioners are trained in the academic context, particularly in undergraduate courses in Brazil and other countries. The authors concluded (i) both national and international scenarios lack time for test practices in the teaching of software testing in software engineering; (ii) the national scene offers few specific software testing disciplines; and (iii) in the international scenario, computer science courses usually offer software testing as a specific discipline.

Ng et al. [12] described software testing practices adopted by the Australian industry. One of the aspects focused is software testing teaching. The research was conducted with 65 professionals more likely to know the environment of the development companies (e.g., senior employees involved with software testing). The authors concluded (i) the lack of knowledge of software testing is the predominant factor for the non-adoption of test activity systematization by organizations, and (ii) testing practitioners are not provided with adequate university formation and enough training when hired by organizations.

Chan et al. [2] conducted a study with industry professionals to understand the tester's education level in the Hong Kong software industry. In particular, the aim was to gain an insight into the extent to which software testing professionals received formal education in the field. A survey conducted with professionals from 34 organizations revealed (i) only 12 organizations had specific software testing teams, and (ii) only 28% of the professionals who were test team members had contact with the test activity during graduation.

Garousi and Zhi [7] conducted a survey with professionals who performed the testing activity in Canadian software organizations. Although the aim was to understand the testing practices used, part of the study focused on the professionals' knowledge in software testing. In particular, the authors asked if they were provided with some formal training in the software testing area. The survey results indicated (i) more than half of the 206 participants had received less than 20 hours of training, and (ii) 39% had no contact with formal test teaching.

Scatalon et al. [17] developed a study in survey format, focused on the gaps observed by Brazilian professionals regarding a testing activity in the software development sector. Among the 90 participants, 74 had learned software testing in the software engineering subject, and only 15 had taken a specific test subject. The major findings were (i) a significant knowledge problem in the topic of web system testing and testing practices in the main testing levels (unit, integration, and systems testing), (ii) a non-wide use of cause-effect graph, finite state machines, control flow graph, data flow analysis and mutation test in practice, and (iii) a most theoretical teaching of software testing to university students.

A particular study was conducted with educators of software engineering and testing, in the national context [15], which investigated the testing contents taught, the teaching methods adopted, the educational resources used in classes, and the teachers' difficulties in teaching the subject. The authors observed (i) functional testing was the testing technique mostly taught by teachers, (ii) teachers devoted efforts towards addressing the different testing levels, (iii) teaching was performed almost exclusively by the traditional model, which prioritizes less practical activities in the classroom, and (iv) most of the educational resources used by teachers were not interactive.

The previous studies do not focus on the teacher's view in the software testing teaching field; the only study that concentrated on this perspective had its context limited to a single country - Brazil.

3 RESEARCH METHOD

This paper addresses an empirical study conducted in the form of a survey that followed the process of [9, 18] for synthesizing the way and the depth at which the topic was addressed in the community of educators and under which teaching practices. The

aim is to provide an overview of software testing teaching from the academic teacher perspective.

3.1 Planning

The survey planning followed the process defined by Kitchenham and Pfleeger [9], and was divided into six main activities:

(1) Definition of the study objectives: identification of the objectives and respective research questions;
(2) Survey planning: construction of the study execution plan;
(3) Development of the pilot study: creation and application of the pilot study for assessing the applicability of the survey;
(4) Pilot study evaluation: correction of possible failures and survey validation;
(5) Survey execution: application of the final version of the questionnaire to the participants for data collection; and
(6) Analysis and discussion of the data: obtaining of data, extraction of relevant information for analyses of the research questions, and discussion of the results.

3.2 Research goals

Among the main objectives of this survey are:

(1) Listing of the contents taught in disciplines that include the software testing topic;
(2) Characterization of teachers' knowledge and experience level;
(3) Description of the teaching approaches employed;
(4) Description of teaching support mechanisms;
(5) Understanding of the way teachers evaluate students' learning; and
(6) Identification of the main challenges faced in the software testing teaching process.

Table 1 shows the research questions that guided the study according to the objectives defined.

Table 1: Research questions.

RQ1. What content is taught in the software testing discipline?
RQ2. What is the teachers'level of knowledge on the topic?
RQ3. What are the approaches used in the teaching process?
RQ4. What support mechanisms assist the teaching-learning process?
RQ5. What assessment tools measure the students' learning?
RQ6. What are the main difficulties in software testing teaching?

3.3 Definition of the Target Population

The target population of this study is professors who teach disciplines that address software testing and related topics in computing courses, Computer Science, Computer Engineering, Software Engineering, or other related courses in higher education institutions worldwide. The survey process for data collection was designed towards reaching as many participants as possible from different nationalities, and the main international conferences in software testing and software engineering area were considered - *International Conference on Software Testing, Verification and Validation* (ICST), *International Symposium on Software Testing and Analysis* (ISSTA), *Joint Meeting on Fundamentals of Software Engineering* (FSE/ESEC), *International Conference on Automated Software Engineering* (ASE), *International Conference on Software Engineering* (ISCE) and *International Conference on Software and Systems Testing* (ICTSS). The authors' email addresses, which totaled 2300, were extracted from the proceedings of the last five years of such conferences and used as the sample composing the target population defined for the survey.

3.4 Survey Design and Administration

The questionnaire considered the research questions produced in the survey planning; it was constructed and made available online[1].

After the design of the instrument, it was administered for a smaller sample of the population (four specialists in the field of software testing who taught the topic in different higher education institutions). This pilot study aimed to verify the construction of the questionnaire, its understanding, and any possible problems faced. Participants suggested improvements in two issues and the necessary adaptations were made by the authors, thus concluding this phase, and assuring reliable assessment and validity of the instrument prior to its application [9].

The survey was sent to the emails obtained in the planning phase in two stages, i.e., a first contact with the participants was made from April 22 to 30, and a reminder was sent from June 15 to July 1. The results are discussed in the following sections.

4 RESULTS

The questionnaire was sent to teachers that worked in the software testing area worldwide; they were identified through articles in conferences in the main areas of software testing and engineering, as explained in Section 3.3. 74 professors answered the questionnaire completely. Table 2 shows the global distribution of the participants.

4.1 Characterization of Participants

As shown in Figure 1, the study involved participants from several regions worldwide, including four of the five continents - Africa was excluded, since no response was received. The European continent represents the highest percentage of respondents (49%), followed by the American continent (32%), Asia (15%), and Oceania (3%).

The Venn diagram shown in Figure 2 represents the undergraduate and graduate courses that address the test topic. The frequency at which the topic is covered in a single course (as part of only one ellipse) or in multiple courses (as part of the intersection of two or more ellipses), considering a participant can actuate in more than one course, can be distinguished. The software testing topic is frequently covered in Computer Science (42.3%), Software Engineering (25.2%), Information Systems (20.3%), and Computer Engineering (8.9%), respectively. Other courses, such as Cybersecurity, Network Engineering, Information Technology, and Mathematical and Computing Science can approach this topic, both with a 0.8% percentage for each course.

Regarding the participant's knowledge level in software testing, as shown in Figure 3, most teachers have academic research knowledge level (54%), followed by teachers with industrial application

[1] <https://forms.gle/4FVzc65qLc5n26AU8>

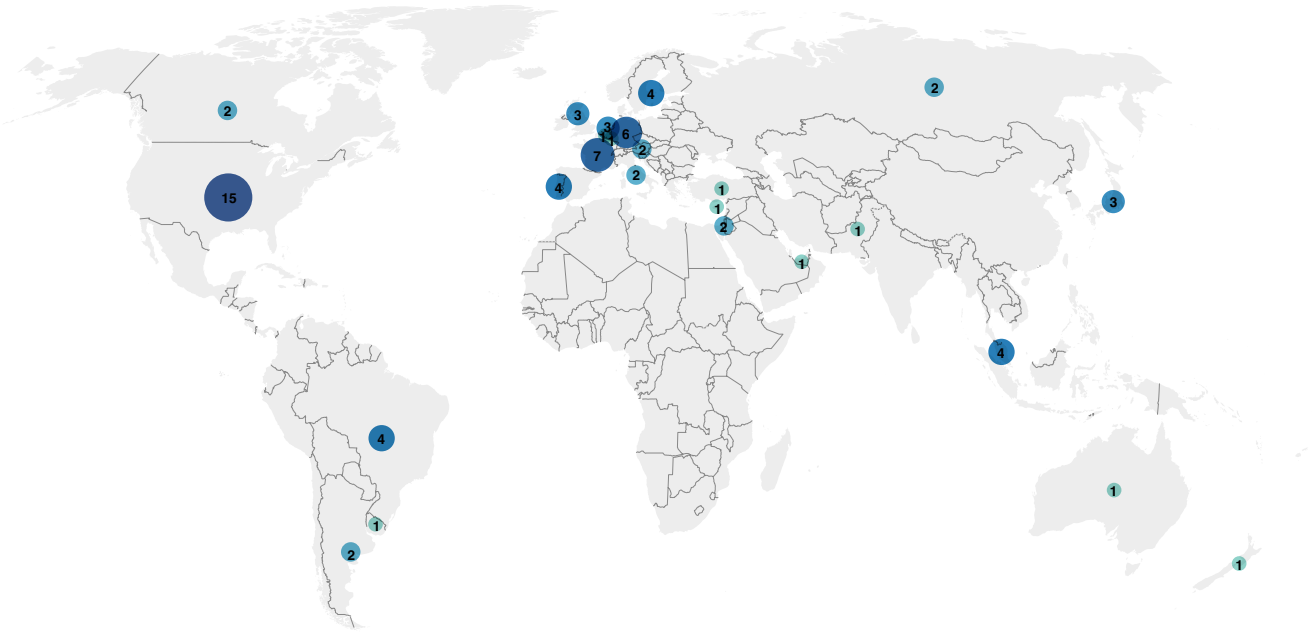

Figure 1: Geographical Distribution of Respondents.

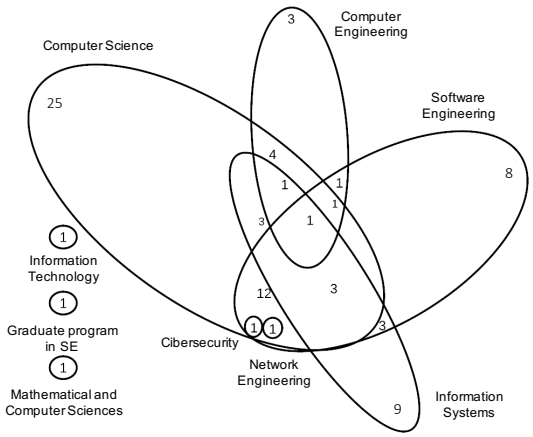

Figure 2: Course that taught the testing topic.

test knowledge level (26%), those with knowledge at classroom application level (19%) and, those with general knowledge level (1%). Among them, 88% have a doctorate and 12% have a master's degree, as shown in Figure 4, which characterizes a diverse sample of the population.

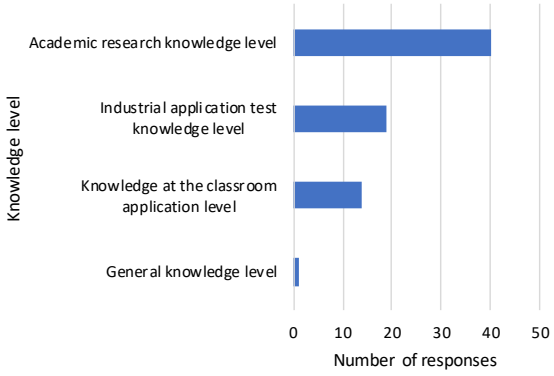

Figure 3: Participants' Knowledge level.

Table 2: Countries where the participants work.

Continent	Country	Participants
America	Argentina	2
	Brazil	4
	Canada	2
	United States	15
	Uruguay	1
Europe	Germany	6
	Austria	2
	Belgium	1
	Cyprus	1
	France	7
	Netherlands	3
	Italy	2
	Luxembourg	1
	Portugal	4
	United Kingdom	3
	Russia	2
	Sweden	4
Asia	United Arab Emirates	1
	Israel	2
	Japan	3
	Malaysia	4
	Pakistan	1
	Turkey	1
Oceania	Australia	1
	New Zealand	1
Total		74

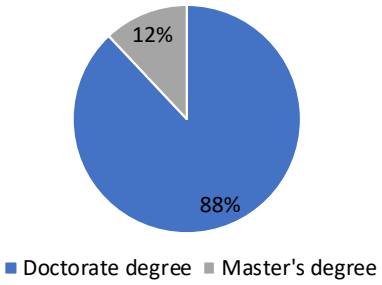

Figure 4: Participants' academic degree.

4.2 Characterization of the Course

Duration, in hours, and type of course (mandatory or elective) were the aspects identified for the characterization of the course that addressed the test topic. Figure 5 shows the course classification in terms of workload - 42% disciplines dealt with software testing 47 hours/class or less, 34% spent from 47 to 88 hours/class, 14% spent from 88 to 129 hours/class, 1% spent 170 to 211, and 1% spent more than 211 hours/class. Regarding education type, 97% of software test courses or courses covered the software test topic face to face, and only 3% addressed it as distance learning/education (see Figure 6).

Since the test topic was addressed in a specific software testing course or covered within a Software Engineering course, in 46.6% of the cases it was taught in a specific testing course, whereas in 25.4%,

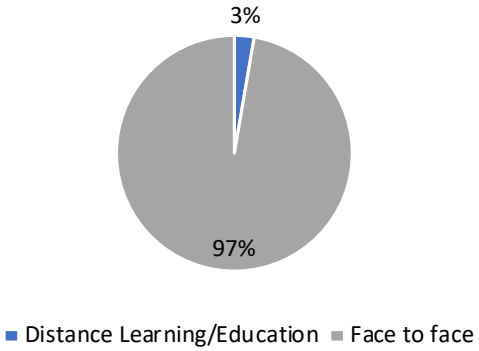

Figure 5: Course Classification.

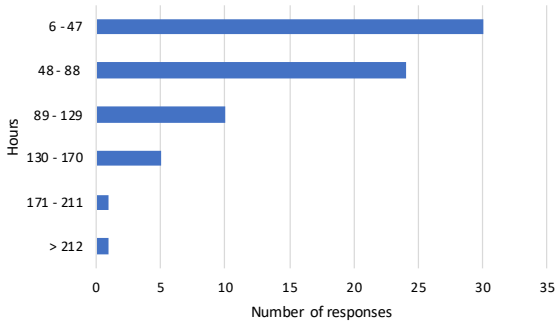

Figure 6: Course Workload.

it was covered in a more general software engineering course, as shown in Table 3.

Table 3: Insertion of the software testing topic in courses.

Course classification	Type	Frequency
Elective	Specific	43.0%
Topic included in a mandatory course	Not specific	20.3%
Topic included in an elective course	Not Specific	5.1%
Mandatory	Specific	3.6%

A question was elaborated for identifying the different nomenclatures used in courses that address software testing (see Table 4 for the results). As shown, the topic has been predominantly explored in software testing Verification & Validation course; however, it can be covered in other courses with different nomenclatures specific to software quality guarantee activities, or in more general Software Engineering courses and other areas, such as Network Engineering, Database Management, Programming, among others.

4.3 Content taught

Specific questions, for the understanding of the way software testing is addressed and the level of details achieved, considering the

Table 4: Course name.

Name	Frequency
Software Testing	28.2%
Software Engineering	22.5%
Software Analysis and Testing	2.8%
Software Quality Engineering	2.8%
Information Systems Engineering	2.8%
Introduction to Software Engineering	2.8%
Software Testing & Verification	2.8%
Software Testing & Verification	2.8%
Development Logic Bases: Modularization, Tests	1.4%
Object Oriented Design	1.4%
Network Engineering	1.4%
Software Engineering II	1.4%
Advanced Software Engineering II	1.4%
Specification, Testing and Verification	1.4%
Quality assurance for IT	1.4%
Generating Software Tests	1.4%
Database Management	1.4%
Principles of Software Engineering	1.4%
Embedded Systems Programming	1.4%
Programming II	1.4%
Information System Design and Development	1.4%
Software Quality	1.4%
Problem Solving II	1.4%
Information Systems	1.4%
Testing Techniques	1.4%
Testing	1.4%
Model-Based Testing	1.4%
Software Testing and Quality Assurance	1.4%
Testing and Validation of Information Systems	1.4%
Software Testing and Validation	1.4%

levels of testing activity, testing techniques and testing criteria covered.

The histogram chart in Figure 7 shows the results, according to the testing level and number of teachers who approached each of such levels in their courses. The most common testing levels in the courses are unit testing (30.6%), integration testing (25.1%), system testing (24.7%), and acceptance testing (17.9%), respectively. Other testing levels are also taught on a smaller scale (e.g., usability, stress, load, and regression testing).

Towards an evaluation of the way the course type (specific or not specific for software testing) influences the content covered, Table 5 shows results related to number of testing techniques covered in the different course types. Functional testing and structural testing rank first, followed by fault-based testing and model-based testing; a few others are covered in specific disciplines. The data clearly show if a course is specific to software testing, the number of different techniques covered is greater in comparison to the more generic courses that address the topic.

Table 6 shows a summary of the main testing techniques and criteria addressed by professors, contrasting the percentage of specific courses that cover software testing and non-specific disciplines, when the test topic is taught as part of the content of another Software Engineering course.

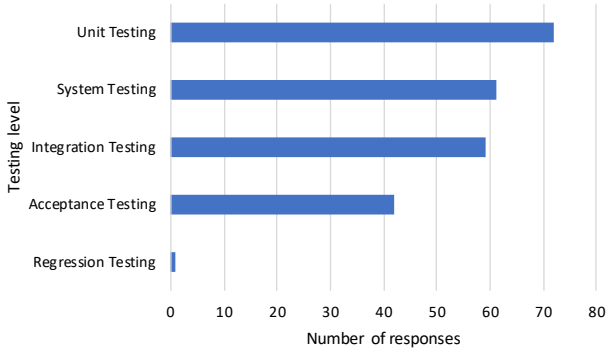

Figure 7: Testing levels taught.

Table 5: Software Testing Techniques.

Testing Technique	Specific Course	Non-Specific Course
Functional Testing	42	24
Structural Testing	38	25
Fault-Based Testing	31	6
Model-Based Testing	27	8
Other	3	0

Table 6: Criteria addressed by testing techniques in specific and non-specific software testing disciplines.

Technique	Test criterion	Specific	%	Non-Specific	%
Functional	Boundary Value Analysis	37	84%	15	50%
	Cause-Effect Graph	10	23%	7	23%
	Equivalence class testing	38	86%	21	70%
	Functional Systematic Testing	18	41%	19	63%
	Combinatorial Testing	2	5%	0	0%
	Domain matrix analysis	1	2%	0	0%
	Use Cases and Risk-based testing	2	5%	0	0%
	None	1	2%	8	27%
Structural	Control-Flow Based Criteria	40	91%	20	67%
	Data-Flow Based Criteria	27	61%	12	40%
	Condition-based testing and MC/DC	1	2%	0	0%
	None	3	7%	7	23%
Fault Based Testing	Mutation testing	34	77%	4	13%
	Error Seeding	17	39%	4	13%
	None	5	11%	23	77%
Model based Testing	Statecharts	15	34%	5	17%
	Finite State Machines	25	57%	5	17%
	UML Diagrams	12	27%	5	17%
	Events Graph	3	7%	1	3%
	Decision Tables	3	7%	0	0%
	None	13	29%	21	70%

The professors who teach in disciplines specific for software testing reported they cover the main functional testing technique criteria, such as equivalence class testing (86%), boundary value analysis (84%), systematic functional testing (41%), and cause-effect

graph (23%). Approximately (5%) address combinatorial interaction testing, domain matrix analysis (2%), and use cases and risk-based testing (5%) - a small portion covers no functional technique criteria (2%). In disciplines non-specific for software testing, the percentage of functional testing criteria covered is lower - equivalence class testing (70%), boundary value analysis (50%), cause-effect graph (23%); it is higher only for systematic functional testing (63%). The remaining criteria covered in specific disciplines (combinatorial interaction testing, use cases and risk-based testing, and domain matrix analysis) are not covered in those non-specific for testing.

Regarding structural software testing technique, among the participants who teach specific software testing disciplines, 91% claim to apply control-flow based criteria and 61% teach data-flow based criteria, 3% address condition-based testing and MC/DC, and 5% do not cover any criterion of the structural testing technique. In contrast, out of teachers who teach non-specific disciplines, 23% address no structural test criteria, 67% teach control-flow-based criteria and 40% teach data-flow-based criteria.

As for fault-based testing technique, the most addressed criteria are mutation testing, covered by 77% of disciplines, error seeding, in 39% of specific software testing disciplines, and only 13% in non-specific disciplines. A large proportion of non-specific discipline teachers (77%) teach no fault-based testing criteria - this number drops to 11% among discipline-specific teachers, which can be attributed to reduced hours for the content in more general software engineering disciplines.

According to the models most used for the generation of test data in model-based testing technique in specific testing disciplines, most approaches use finite state machines (57%), followed by statecharts (34%), UML diagrams (27%), events graph (7%) and decision tables (7%). In non-specific disciplines, each model is taught at an equal frequency (17%), except for those that use events graph, which is taught by 3% of the participants, and decision tables, which are not addressed. Still considering the type of course (specific or not specific for the test topic), 29% of the participants who teach in a specific discipline reported that do not cover any type of model-based technique, while in non-specific disciplines, this number reaches 70%.

In general, most participants reported applying different testing techniques and criteria when the discipline is specific to software testing. However, some criteria are not approached, even if the discipline is specific. This fact can be recurrent in the limited time devoted to the discipline, and also due to the diversity of test techniques, which requires prioritization by the teacher. Other disciplines also share a workload with other topics, varying according to the teachers' focus on the interdisciplinarity of the topic and the wide range of related themes.

Towards including other possible topics covered in software testing disciplines, participants were encouraged to list those not addressed in our questionnaire. Table 7 shows such topics and the frequency at which they were mentioned by the participants, who cited mostly basic concepts and terminology (90%) in specific and non-specific disciplines, software testing tools, of which 93% were addressed in specific disciplines, and 63% were taught in non-specific ones. Professors in specific disciplines reported software testing process (70%), automated software test data generation (61%), verification and validation activities (59%), software testing documentation (50%), software debugging (48%), test automation (5%), testing smells (5%) and TDD (2%), in a decreasing order of frequency. In non-specific software testing disciplines, 63% of teachers used testing tools, 57% addressed verification and validation activities and software testing process, 33% covered software debugging, 26% taught documentation, and 17%, 7% and 3% addressed automated software test data generation, TDD and test automation, respectively.

Table 7: Other software testing topics covered.

Other Topics Addressed	Specific	%	Non-Specific	%
Basic concepts and Terminology	40	91%	27	90%
Software Debugging	21	48%	10	33%
Testing documentation	22	50%	8	26%
Software testing tools	41	93%	19	63%
Automated software test data generation	27	61%	5	17%
Verification and validation activities	26	59%	17	57%
Software testing process	31	70%	17	57%
TDD	1	2%	2	7%
Test automation	2	5%	1	3%
Test Smells	2	5%	0	0%

4.4 Teaching Approaches and Educational Resources

This study also aimed at identifying the teaching approaches adopted in software testing disciplines. According to the results in Table 8, the traditional teaching approach is still the most used (73% of the participants, considering each of them used more than one teaching approach). Problem-based learning reached 66%, and project-based learning reached 46%. On a smaller scale are joint test teaching with programming 19%, flipped-classroom, and peer reviews, both with 9%. Participants also reported using workshops, and challenges (1% each).

Table 8: Teaching Approaches.

Approach	Participants	%
Traditional	54	73%
Project-based learning	50	66%
Problem-based learning	34	46%
Joint test teaching with programming	14	19%
Flipped Classroom	7	9%
Peer review	7	9%
Workshops	1	1%
Challenges	1	1%

The educational resources most used for supporting software testing teaching are listed in Table 9. Presentations in slides format achieved above 80% frequency, followed by book chapters (63%),

scientific articles (49%), tutorials (41%), questionnaires (39%), videos (32%), and learning management system (30%). A significant number of participants uses educational modules (16%), educational games (10%), source codes (5%), and massive open online courses - MOOCs (4%), and some use social media, podcasts and jupyter notebooks [2] (3% each).

Table 9: Educational Resources.

Resource	Participants	%
Slides	62	84%
Book Chapters	47	63%
Scientific articles	36	49%
Tutorials	30	41%
Quizzes	28	39%
Videos	24	32%
Learning management systems	22	30%
Educational modules	12	16%
Educational games	7	10%
Source Code	4	5%
MOOCs	3	4%
Jupyter Notebooks	2	3%
Podcasts	2	3%
Social Networks	2	3%

4.5 Instruments for knowledge measurements

Table 10 shows the instruments used by the participants to measure students' knowledge (each participant could use more than one instrument). Practical works (73%), followed by practical tests (66%), multiple choice tests (40%), discursive tests (23%), oral tests (11%) and research work (12%), besides homework (3%) and projects-based problems (1%), were cited. This result differs from the teaching approaches used showing that, although using the traditional teaching approach, educators apply practical instruments to measure students' learning.

Table 10: Assessment instruments.

Assessment resource	Participants	%
Practical works	54	73%
Practical tests	49	66%
Multiple Choice tests	30	40%
Discursive tests	17	23%
Research work	9	12%
Oral tests	8	11%
Homework	2	3%
Projects-based problems	1	1%

[2] <https://jupyter.org>. Jupyter Notebooks project provides applications on the web allowing creation and sharing of documents containing active codes, equations, visualizations, and narrative text in various computing areas, such as numerical simulation, statistical modeling, data responses, machine tests, among others.

4.6 Difficulties in Teaching Software Testing

An open discursive question was available for the participants to report the difficulties most commonly faced. Figure 8 displays the results, as well as categories of difficulties and the proportion in which they were cited.

Table 11 displays the difficulties, of which the main one is related to stimulating students' interest in the software testing discipline, which, in most cases, is elective, according to Table 3. Reduced class time, as well as lack of students' interest, lack of practical examples (which represent real industry projects), and distance between content taught and practices in the industry were also reported.

Teachers also face problems with the division of content in the discipline planning in reduced time, and, in many cases, in more general disciplines of software engineering. The lack of supporting testing tools also hinders the learning process, making the topic complex and reducing students' motivation, who often lack basic knowledge to develop satisfactory projects as practical activities. Moreover, since course coordinators generally consider the topic of a secondary subject in their curricula, they do not encourage projects or recover funds for the acquisition of technologies.

Previous literature reviews [6, 21] also evidenced such difficulties, especially lack of students' motivation, lack of available testing tools, difficult rationalization of contents, and distance between academia and industry. Other new difficulties have been identified in our research, which is important for both understanding of the state-of-the-art in the area and development of research towards possible solutions to those challenges.

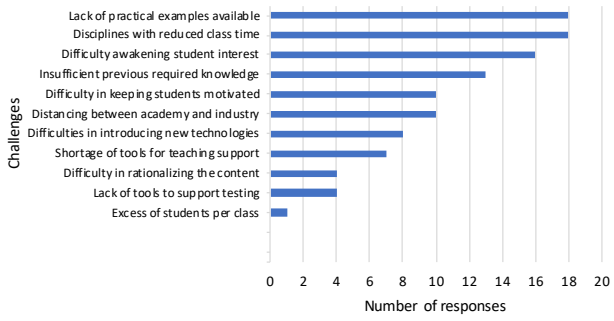

Figure 8: Challenges in Software Testing Teaching.

5 DISCUSSION

The analysis of research questions revealed the way software testing has been taught worldwide. Below is a summary of the main results.

The topics and depth at which they are taught have shown a significant difference, according to the classification of the disciplines (specific of software testing or taught as part of a more general one in software quality or engineering). The main techniques reported are functional and fault-based testing, examined in most disciplines, however, a variety of testing criteria for each of them are still restricted to specific disciplines.

Table 11: Difficulties Listed by the Participants.

Difficulties	Description
Difficulty in awakening students' interest	Difficulty in awakening student interest in test topic and showing the real importance of this activity in the software development industry.
Shortage of tools for teaching support	Low availability of tools, such as educational games, online courses, questionnaires, and others, which might help students understand the content.
Insufficient previous knowledge of students	The lack of students' previous knowledge in programming hampers the understanding of the software artifact under testing, hence, the testing activity.
Distancing between academia and industry	Gap between what industry demands and what is addressed at the university.
Rationalization of contents by teachers	Distinction between topics and their most important subtopics, requirements applicable in the classroom, and sequence of application.
Insertion of new technologies in the curricula	Low incentives from the university, which results in a lack of research projects and investments in new technologies for assisting disciplines.
Keeping students' motivation	Complexity and specificity of the testing area, which reduce students' interest – students concentrate their efforts on mandatory subjects of the course core.
Disciplines with reduced class time	Smaller number of testing techniques and criteria applied when software testing is covered in a more general discipline.
Lack of tools for supporting testing	Few tools that automate the test application and difficult implementation of testing techniques.
Lack of practical examples available	Low availability of open license examples and industry examples that are in line with students level of development.
Excess of students per class	Teachers cannot satisfactorily devote attention to all students.

The most explored teaching approaches are project-based and problem-based learning, and the main educational resources are slides, book chapters, tutorials, quizzes, videos, and learning management systems. Other teaching approaches and educational resources are used, but less frequently.

Teaching assessment is undertaken mainly through practical work and practical tests; it has been applied to several courses, in multiple-choice and discursive tests. Research articles, oral exams, homework, and problem-based projects are less frequently employed.

Among the main difficulties raised by teachers are reduced time for the development of the topic, need for alignment between academia and industry, discipline classification (whether elective or mandatory), lack of testing support tools, lack of students' basic skills, and students' motivation and engagement.

The results provide an overview of the area from a teachers' perspective, and a basis for both confrontation between literature and real teaching experiences, towards confirming or refuting hypotheses, and new research lines in this field.

6 THREATS TO VALIDITY

- **Internal validity**: the respondents might interpret the research questions differently from the researchers' intentions. Towards mitigating this threat, each question provides a help resource that details its purpose. Although it may not guarantee the total exclusion of the threat, it can improve the respondents' understanding.
- **Construction validity**: towards mitigating construction problems in our survey, we designed a pilot study containing a preliminary questionnaire that was sent to a group of software testing specialists; they reviewed the survey content and indicated some problems, which were fixed.
- **Conclusion validity**: the analyses of results may be incorrect due to researchers' interpretation problems. Therefore, descriptive statistics techniques analyze the results for each research question.
- **External validity**: this threat refers to the non-replicability of the study. Towards mitigating it, both the survey and the list of conferences used for the collection of participants' contacts have been made available. The representativeness of the population sample cannot be guaranteed; the difficulty in reaching all educators and, especially, ensuring all would respond the questionnaire is a recurring problem in empirical studies of this nature.

7 CONCLUSIONS

This paper has reported results of an empirical study designed as an opinion survey conducted with educators in software testing from worldwide universities.

Some similarities on the way software testing has been taught from a national to a global scale have been observed. Unit testing, for example, is the most testing level taught both in Brazil and in the international context.

Regarding the content covered in disciplines, functional test (partitioning into equivalence classes and boundary value analysis criteria) is the mostly taught, followed by structural testing and control-flow based criterion, and then data flow-based criterion. Mutation test is the most taught criterion in fault-based testing technique. Some differences were observed only for model-based testing, according to the approach used for the generation of test data. In Brazil, the most used approach is the generation of test sequences

followed by finite state machines, respectively. In a worldwide context, the range of criteria is broader and the most used approach is the one that uses finite state machines, followed by statecharts, UML diagrams, events graphs, and decision tables, respectively.

In both contexts, the traditional approach is mostly used, followed by project-based learning. The educational resources mostly applied by educators are slides and book chapters. In international universities, the major assessment instruments are practical works followed by practical tests, whereas Brazilian universities employ practical works followed by discursive tests.

Regarding challenges in Brazilian courses, the main difficulty is the lack of time for the teaching of contents, since, in some cases, the courses do not offer a specific discipline. In a global context, educators reported their difficulty in keeping students interested in software testing. Although clear, many difficulties faced are the same, differing only in the frequency of occurrence.

Some differences were detected in the characterization of the participants and disciplines. Regarding the workload of software testing disciplines, it is approximately 60 hours in Brazilian universities and 68 hours worldwide. 82% of the professors who teach the discipline, in an international context, have a doctorate degree, and 12% are masters; in the national context, 61% have a doctorate degree, 37% are masters, and 2% are specialists. However, in both contexts, most teachers have an academic research knowledge level.

The article has provided an overview of the software testing topics covered by disciplines, which can be used as a guide for helping new teachers in their planning of disciplines. The main learning tools used have been described and categorized.

The teaching methodologies applied and problems faced can be useful for new research into solutions, improving the software testing teaching-learning process.

As future work, we aim at comparing our results with those from other studies on software testing teaching, and investigating the same research questions in industrial software testing training environments.

ACKNOWLEDGMENTS

This research was supported by PROPP/UFGD (SigProj number: 329332.1866.8276.06042019 and 322855.1174.8276.11032019), FAPESP (2018/26636-2 and 2019/06937-0), CNPq (312922/2018-3) and CAPES - 001.

REFERENCES

[1] T. Astigarraga, E. M. Dow, C. Lara, R. Prewitt, and M. R. Ward. 2010. The Emerging Role of Software Testing in Curricula. In *IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments*. 1–26.

[2] F. T. Chan, T. H. Tse, W. H. Tang, and T. Y. Chen. 2005. Software testing education and training in Hong Kong. In *International Conference on Quality Software*. 313–316.

[3] P. J. Clarke, D. Davis, T. M. King, J. Pava, and E. L. Jones. 2014. Integrating Testing into Software Engineering Courses Supported by a Collaborative Learning Environment. *ACM Transactions on Computing Education* 14, 3 (2014), 18:1–18:33.

[4] A. C. Dias-Neto, S. Matalonga, M. Solari, G. Robiolo, and G. H. Travassos. 2017. Toward the characterization of software testing practices in South America: looking at Brazil and Uruguay. *Software Quality Journal* 25, 4 (2017), 1145–1183.

[5] V. Garousi and A. Mathur. 2010. Current State of the Software Testing Education in North American Academia and Some Recommendations for the New Educators. In *IEEE Conference on Software Engineering Education and Training*. 89–96.

[6] V. Garousi, A. Rainer, P. Lauvås Jr., and A. Arcuri. 2020. Software-testing education: A systematic literature mapping. *Journal of Systems and Software* 165 (2020), 110–570.

[7] V. Garousi and J. Zhi. 2013. A survey of software testing practices in Canada. *Journal of Systems and Software* 86, 5 (2013), 1354 – 1376.

[8] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale. 2018. Software testing: Survey of the industry practices. In *International Convention on Information and Communication Technology, Electronics and Microelectronics*. 1449–1454.

[9] B. A. Kitchenham and S. L. Pfleeger. 2008. Personal opinion surveys. In *Guide to advanced empirical software engineering*. Springer, New York, NY, 63–92.

[10] O. A. L. Lemos, F. F. Silveira, F. C. Ferrari, and A. Garcia. 2017. The impact of Software Testing education on code reliability: An empirical assessment. *Journal of Systems and Software* (2017), 497–511.

[11] G. J. Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing*. Wiley Publishing, Hoboken, Nova Jersey, EUA.

[12] S. P. Ng, T. Murnane, K. Reed, D. Grant, and T. Y. Chen. 2004. A preliminary survey on software testing practices in Australia. In *Australian Software Engineering Conference*. 116–125.

[13] L. N. Paschoal, B. R. N. Oliveira, E. Y. Nakagawa, and S. R. S. Souza. 2019. Can We Use the Flipped Classroom Model to Teach Black-Box Testing to Computer Students?. In *Brazilian Symposium on Software Quality*. 158–167.

[14] L. N. Paschoal, L. R. Silva, and S. R. S. Souza. 2017. Abordagem Flipped Classroom em Comparação com o Modelo Tradicional de Ensino: uma Investigação Empírica no Âmbito de Teste de Software. In *Simpósio Brasileiro de Informática na Educação*. 476–485.

[15] L. N. Paschoal and S. R. S. Souza. 2018. A survey on software testing education in brazil. In *Brazilian Symposium on Software Quality*. 334–343.

[16] L. N. Paschoal, L. F. Turci, T. U. Conte, and S. R. S. Souza. 2019. Towards a Conversational Agent to Support the Software Testing Education. In *Brazilian Symposium on Software Engineering*. 57–66.

[17] L. P. Scatalon, M. L. Fioravanti, J. M. Prates, R. E. Garcia, and E. F. Barbosa. 2018. A survey on graduates' curriculum-based knowledge gaps in software testing. In *IEEE Frontiers in Education Conference*. 1–8.

[18] F. Shull, J. Singer, and D. I. K. Sjberg. 2010. *Guide to Advanced Empirical Software Engineering* (1st ed.). Springer Publishing Company, Incorporated, New York, NY.

[19] A. Soska, J. Mottok, and C. Wolff. 2016. An experimental card game for software testing: Development, design and evaluation of a physical card game to deepen the knowledge of students in academic software testing education. In *IEEE Global Engineering Education Conference*. 576–584.

[20] G. Tan. 2016. A Collection of Well-Known Software Failures. Available at http://http://www.cse.psu.edu/~gxt29/bug/softwarebug.html. Penn State University. Pennsylvania, PA, USA. (2016).

[21] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado. 2015. CS curricula of the most relevant universities in Brazil and abroad: Perspective of software testing education. In *International Symposium on Computers in Education*. 62–68.

[22] M. Zhivich and R. K. Cunningham. 2009. The Real Cost of Software Errors. *IEEE Security Privacy* 7, 2 (2009), 87–90.