Teachers' Perspective on Software Testing Education

ANNA RITA FASOLINO, University of Naples Federico II, Italy
BEATRIZ MARÍN, Universitat Politècnica de València, Spain
TANJA E. J. VOS, Universitat Politècnica de València, Spain and Open Universiteit, , The Netherlands
ALEXANDRA MENDES, INESC TEC, Faculty of Engineering, University of Porto, Portugal
ANA C. R. PAIVA, INESC TEC, Faculty of Engineering, University of Porto, Portugal
FELIX CAMMAERTS, KU Leuven, Belgium
MONIQUE SNOECK, KU Leuven, Belgium
MEHRDAD SAADATMAND, RISE, Sweden
PORFIRIO TRAMONTANA, University of Naples Federico II, Italy

Context: Software testing is a critical aspect of the software development lifecycle, yet it remains underrepresented in academic curricula. Despite advances in pedagogical practices and increased attention from the academic community, challenges persist in effectively teaching software testing. Understanding these challenges from the teachers' perspective is crucial to aligning education with industry needs.

Objective: To analyze the characteristics, practices, tools, and challenges of software testing courses in higher education, from the perspective of educators, and to assess the integration of recent pedagogical approaches in software testing education.

Method: A structured survey consisting of 52 questions was distributed to 143 software testing educators across Western European universities, resulting in 49 valid responses. The survey explored topics taught, course organization, teaching practices, tools and materials used, gamification approaches, and teacher satisfaction.

Results: The survey revealed significant variability in course content, structure, and teaching methods. Most dedicated software testing courses are offered at the master's level and are elective, whereas testing is introduced earlier in less specialized (NST) courses. There is low adoption of formal guidelines (e.g., ACM, SWEBOK), limited integration of non-functional testing types, and a high diversity in textbooks and tools used. While modern practices like Test-Driven Development and automated assessment are increasingly adopted, gamification and active learning approaches remain underutilized. Teachers expressed a need for improved and more consistent teaching materials.

Conclusion: The study highlights a mismatch between academic practices and industry expectations in software testing education. Greater integration of standardized curricula, broader adoption of modern teaching tools, and increased support for teachers through high-quality, adaptable teaching materials are needed to enhance the effectiveness of software testing education.

Authors' Contact Information: Anna Rita Fasolino, fasolino@unina.it, University of Naples Federico II, Naples, Italy; Beatriz Marín, bmarin@dsic.upv.es, Universitat Politècnica de València, València, Spain; Tanja E. J. Vos, tvos@dsic.upv.es, Universitat Politècnica de València, València, Spain and Open Universiteit, , The Netherlands; Alexandra Mendes, alexandra@archimendes.com, INESC TEC, Faculty of Engineering, University of Porto , Porto, Portugal; Ana C. R. Paiva, apaiva@fe.up.pt, INESC TEC, Faculty of Engineering, University of Porto, Porto, Portugal; Felix Cammaerts, felix.cammaerts@kuleuven.be, KU Leuven, Leuven, Belgium; Monique Snoeck, monique.snoeck@kuleuven.be, KU Leuven, Leuven, Belgium; Mehrdad Saadatmand, mehrdad.saadatmand@ri.se, RISE, Vasteras, Sweden; Porfirio Tramontana, ptramont@unina.it, University of Naples Federico II, Naples, Italy.

Manuscript submitted to ACM

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **Applied computing → Education**; • **General and reference → Surveys and overviews**.

Additional Key Words and Phrases: Software Testing Education, Teachers' perspective on software testing education, Software Testing Education Practices, Teachers' Survey

ACM Reference Format:
Anna Rita Fasolino, Beatriz Marín, Tanja E. J. Vos, Alexandra Mendes, Ana C. R. Paiva, Felix Cammaerts, Monique Snoeck, Mehrdad Saadatmand, and Porfirio Tramontana. 2025. Teachers' Perspective on Software Testing Education. 1, 1 (November 2025), 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Although software testing is essential for evaluating the quality of the final product, it often does not receive the attention it deserves. This is due to the absence of a robust testing culture in IT and the lack of emphasis on software testing in IT training, resulting in a shortage of skilled professionals in the testing industry.

Recently, the academic and scientific communities have paid increasing attention to software testing education by launching various initiatives. The official ACM recommendations for CS, CE, and SE curricula, as well as the SWEBOK [78], now devote more space to the different themes and aspects of software testing. These recommendations strive to suggest essential topics for training future IT professionals. On the other hand, the scientific community has proposed a multitude of innovative pedagogical approaches, teaching practices, techniques, and tools to effectively support learning software testing [43] [83] [62].

However, despite these efforts, there are still considerable challenges in teaching software testing [6]. Some examples of challenges that still need to be addressed in this field include bridging the gap between theory and practice, taking into account the real learning needs of students, finding solutions to engage them in learning software testing, and preparing them for their future careers by considering what is really needed by the industry.

To increase attention devoted to software testing and address this gap, it is essential to tackle the problem at its root: the educational field. Specifically, a clear understanding of the current state of software testing education at various levels is needed to better identify the gaps that must be addressed to integrate theory and practice.

When discussing education, two important viewpoints must be considered: those of students pursuing computer science or related degrees and those of teachers instructing these courses. From a general perspective, student engagement (e.g. with online course material [77] or in lab sessions [71]) holds high predictive power on students' academic success, as do course designs that target student motivation. Prior research investigating students' needs in the context of software testing suggests some possible changes to current teaching practices. For example, software testing students desire more engagement through gamification, as well as diversification of teaching practices and tools [17]. Doorn et al.'s observation of students' test case design approaches suggests that shifting from a rationalism-based paradigm to an empiricism-based one is necessary to improve students' software testing skills [27].

The software testing teacher's perspective has been investigated both through local [58] and global surveys conducted among researchers in the software testing field [56] and through a direct analysis of course curricula which were classified according to testing topic taught, teaching approach used, courseware and other organizational aspects [9], [76], [32], [12], [8], [45], [74]. Of particular interest is the study of Ardic et al. [8] that identifies gaps between education and industry needs such as acceptance and security testing, as well as the study of Tramontana et al. [74] showing that software testing courses are most often taught only at the master's level and projects are used as (partial) assessment

method in only 60% of courses. However, the aforementioned analyses limited their data collection to public course pages, which typically only provide generic information, may not represent current teaching practices, and often lack details on the educational tools used.

The aim of our research is to address the gap left by prior research on software testing education and to contribute to the knowledge gaps in the following way:

- Surveying testing course instructors directly. Unlike the study by Melo et al. [56], which distributed its survey globally to researchers– many of whom were not necessarily instructors– selected based on their authorship of software testing conference papers, our survey specifically targeted software testing educators in higher education, focusing on institutions in Western Europe.
- Gain deeper insights by collecting detailed information structured around the framework of teaching practices and recommendations derived from the literature [43], including testing topics covered, teaching approaches, learning objectives, adopted testing materials, and strategies for learning and evaluation.
- Collect the main issues, needs, and challenges experienced by teachers in teaching software testing. This perspective has not yet been addressed in prior surveys.

The survey was released over the period from May 2023 to February 2024 to a total of 143 teachers and received 49 valid responses. The survey questions and the dataset of the collected answers are made publicly available[1].

The remainder of this paper presents the survey study including its design, distribution, collected data and findings. Section 2 reviews related work on secondary studies in software testing education and applied teaching practices. Section 3 outlines the research questions and describes the steps taken in designing the survey, leading to the final set of survey questions. Section 4 reports the survey results, while Section 5 analyzes their relation to recent literature. Threats to validity are addressed in Section 6, and Section 7 concludes the paper with future research directions.

2 Related Works

For the related work of this paper we will consider three different sources. On the one hand we are interested in understanding the current landscape of software testing education research. To do this we consider (1) secondary studies on software testing education which have been published in recent years (subsection 2.1). These papers allow us to identify challenges in the field and proposed approaches to tackle those. Based on a subset of these secondary studies we then take a deeper dive into (2) teaching practices which have been proposed in the literature (subsection 2.2), as they are of particular interest for our paper with its focus on the integration between theory and practice. On the other hand we are interested in understanding the teaching practices applied within current curricula for which we identified papers that (3) investigated software testing courses curricula (subsection 2.3). This allows us to understand the discrepancy between theory and practice in software testing education.

2.1 Secondary Studies in Software Testing Education

In recent years, a number of literature review studies have been published that specifically address the teaching of software testing.

Scatalon et al. [61] synthesized the challenges of integrating software testing into introductory programming courses, as reported by 158 papers, which included: (1) Determining how programming and testing should be connected and delivered together; (2) Dealing with students who do not appreciate the value of software testing; (3) Determining

[1]Anoymized results from the survey, https://anonymous.4open.science/r/TeacherSurveyStudy-2B1D/

how the testing activity should be conducted in programming assignments; (4) How to help students become better testers; and (5) Choosing appropriate tools. The study also discussed possible solutions to the challenges addressed in the literature.

Other systematic mapping studies examined the use of gamification in software engineering education and reported their application in the context of software testing. Pedreira et al. in 2015 [59] reviewed the main gamification approaches in the literature related to software engineering activities. They identified 29 primary studies published up to 2014, of which only three focused specifically on gamification in software testing.

A more recent contribution on the same topic is provided by De Jesus et al. [23]. In 2021, they identified 15 different studies proposing and experimenting with techniques and tools for software testing gamification in the context of academic courses. The authors classified these studies according to the involved testing activities, levels, techniques, tools, and the gamification goals and approaches employed.

The most recent and comprehensive secondary study on software testing education has been conducted in 2020 by Garousi et al. [43]. This work presents a systematic literature mapping (SLM) that synthesizes the research published by the education community between 1992 and 2019. Analyzing a pool of 204 scientific papers, the study reveals that numerous pedagogical approaches, courseware, and specific tools for teaching software testing have been proposed in the literature. We discuss some of the identified pedagogical approaches of this paper in more detail in the next section. In addition, Garousi et al. showed that many challenges in testing education exists and provides insights on how to possibly overcome those challenges. From the student perspective, (1) testing is often not well accepted by students and perceived as boring, (2) students may face tool-specific challenges that can be overwhelming and (3) learning software testing adds to their cognitive load. From the instructor perspective, (1) time and resource constraints often limit the opportunity to teach testing effectively within programming courses, and (2) evaluating students' test cases frequently requires substantial manual effort, making assessment challenging and time-consuming. While Garousi et al. [43] provided a high-level overview of software testing education, they did not investigate the practical application of these research findings in actual university courses from the instructors' point of view.

These secondary studies give an overview of pedagogical approaches and student-centered challenges, but lack systematic data on the organizational characteristics and structural aspects of actual software testing courses. While Garousi et al. [43] analyzed 204 papers on testing education, the focus remained on proposed approaches rather than empirical data about course organization, credit allocation, or enrollment patterns. This gap informed our survey design, particularly RQ1 (organizational characteristics) and RQ6 (teacher satisfaction), enabling us to gather systematic data about the practical realities of course delivery from the instructor perspective.

2.2 Teaching Practices for supporting Testing Education

The work of Garousi et al. [43] provides a fundamental starting point for analyzing the state of the art in the education in software testing described in the literature. This study shows that the top three types of contributions to software-testing education made by the papers in this area are: Pedagogical approaches, Proposing a specific tool for testing education, and Courseware / new course proposal. These are discussed in more detail in the following paragraphs.

2.2.1 Pedagogical approaches. Within the Pedagogical approaches category, the SLM extracted a number of beneficial teaching practices that aim to overcome well-known issues and challenges in software testing education. One recommended practice is to introduce testing and TDD early in introductory programming courses [24, 29] in order to help students develop a correct mental model about software testing from the outset. Another practice involves the use of

free, open-source, real-world projects [52], or software written by fellow students [20], to increase students' motivation by encouraging them to find and fix other people's bugs rather than their own. A further practice, frequently reported in the analyzed studies, is the adoption of active learning approaches [44], as opposed to traditional lecture-based methods. Active learning provides students with opportunities to think critically about ideas through activities that both deepen and challenge their understanding. It encourages them to engage actively with course materials, peers, and instructors, in contrast to passive learning approaches that rely primarily on reading or listening to a "talking head" style of instruction. According to Cattaneo [19], active learning activities can be classified into problem-based, discovery-based, inquiry-based, project-based, and case-based learning.

2.2.2 Educational tools. Educational tools designed to support students in the learning process have also been widely reported in the literature. Such tools can provide learners with practical experience and foster the acquisition of testing techniques in engaging and motivating ways.

Interactive learning. A first category of educational tools are the ones implementing an environment to guide students through a learning path, interactively, or with the support of machine learning techniques. As an example, when TDD is used in an introductory course, WebIDE can guide the students through a pre-defined path of steps in order to force them to write tests and specifications before implementing solutions [28]. Another interesting example is that of an automated and interactive system designed to help students learn how to write better test cases, thanks to the feedback they receive on their test cases [66]. The system gives clear examples of buggy implementations that their tests do not detect.

Supported learning. Another category is that of tools offering facilities to support students in practicing testing in laboratory settings. For example, Weikle et al. [79] proposed a framework that automatically does unit testing and grading for an intermediate level systems course project. Similarly, Wen et al. [80] have proposed a software test laboratory on a cloud platform. This platform provides rich functionalities such as developer testing.

Gamification. Finally, there are tools that exploit a gameful approach to better motivate students to practice testing. Gamification has been defined as «the use of game design elements in non-game contexts» [25]. It seems to be a promising approach to improve the experience of (learning about) testing, by trying to make testing a "fun" activity. Two main categories of tools that exploit gamification have been proposed in the field of software testing education. The former includes tools that implement a game with specific game dynamics, mechanics, and components that try to stimulate students by means of challenges and competitions. They include typical gamification elements such as reward points, badges, avatars, and leader boards in a learning environment. The latter includes tools that offer a simulation/ education game where students can practice testing on small/ toy examples. Bug Hunt, for instance, is a web-based environment that contains four introductory lessons on testing terminology, black and white box techniques, and testing automation and efficiency [30]. It incorporates challenges in each lesson and provides immediate feedback to promote engagement while students practice the application of fundamental testing techniques. It provides a complete and automatic assessment of student performance to reduce the instructor's load. Other examples include HALO (Highly Addictive sociaLly Optimized Software Engineering) [14], Code Defenders [35], Testing Game [47] and GAMFLEW [65].

2.2.3 Courseware. Among the various courseware materials used in software testing education, those designed for assessment play a particularly important role. Assessing the students' work is a challenging activity for teachers, as

reported in the literature [43]. [63] noted that current assessment techniques based on automated grading tools for evaluating student-written software tests are imperfect. [73] observes that grading exercises requires substantial effort, effort that could otherwise be devoted to supporting the learning process. Among existing assessment practices, a common approach is to check the code coverage of the implemented tests. But code coverage alone is not a satisfactory measure for test quality [46]. Advanced testing techniques like mutation testing [50] could help in such regard, but, unfortunately, it is not so well known outside of test specialists, and tool support is still rather unsatisfactory [3]. Other studies suggest adopting automated assessment approaches embedded in learning environments like Web-CAT [2], or incorporating self and peer-assessment strategies to complement traditional evaluation methods.[7] .

In summary, the literature identifies specific evidence-based practices that have shown promise in small-scale studies, yet their widespread adoption remains unknown. The detailed categorization of pedagogical approaches, educational tools, and courseware directly informed our survey structure, particularly RQ3 (teaching practices), RQ4 (educational tools), and RQ5 (gamification approaches).

2.3 Current state of Software Testing Curricula

Several prior works have examined the state of software testing education within computer science and related curricula.

In the United States, Astigarraga et al. [9] analyzed the course offering of 27 highly ranked computer science programs in 2010. They found that fewer than 5% of those programs offered courses dedicated to software testing. Moreover, the study revealed that such courses are primarily available at the graduate and research levels. The authors suggest that this limited availability may be due to the relatively low level of professional software testing experience among the faculty in these programs.

In Brazil, Valle et al. [76] analyzed the course offerings of 25 highly ranked universities in 2015, focusing on programs in computer science, computer engineering, information systems, and software engineering. They found that, in Brazilian universities, courses addressing verification, validation, and software testing are generally offered with little integration with other disciplines. Most of these courses allocate less than ten hours to software testing, which the authors consider insufficient for developing a solid understanding of the topic. Moreover, only a few universities were found to offer dedicated software testing courses. These dedicated courses typically approach testing methodologically but lack integration with related subjects, such as data structures and databases. Extending their analysis to institutions abroad, including in the UK and Switzerland, the authors observed similar trends.

Another study in Brazil, conducted by Elgrably et al. [32], analyzed 28 Brazilian institutions in 2015, expanding upon Valle et al.'s [76] selection to provide a more in-depth examination of the Brazilian software testing education landscape. The study found that 32% of the institutions did not offer dedicated software testing courses. Testing techniques and basic test concepts were the topics most frequently covered, and software engineering was the course that most often included some software testing content. The authors also compared the prevalence of these topics with the ACM/IEEE curriculum guidelines, finding that black-box testing was the most thoroughly covered topic, whereas exception handling was the least integrated.

On a global scale, Melo et al. [56] conducted a survey in 2010 targeting 74 professors with questions concerning topics taught, teacher's level of knowledge on the topics, used teaching approaches etc. They found that project-based and problem-based learning were the most commonly used teaching approaches, with evaluations primarily conducted through practical work and tests. The study also highlighted a key challenge for instructors: the need to better align educational content and practices with industry requirements. It should be noted that the data, collected in 2010, may

not reflect recent developments, and that the surveyed teachers were chosen based on their authorship of software testing conference papers.

Ardic et al. [8] investigated the current state of software testing education in 2023, analyzing the curricula of 83 courses from the top 100 universities according to the Times Higher Education ranking. The study found that about 65% of the dedicated testing courses covered topics related to "test process" and "test type". Additionally, based on a survey of 51 practitioners, the researchers identified a general need for more knowledge in many testing areas, including acceptance testing and security testing. They suggest that this gap may partially result from higher education programs not adequately preparing students for those types of testing.

In 2024, Hanna [45] analyzed the topics covered by 80 software testing courses from nine different countries and compared them with the skills sought by 400 software testing related job advertisements. The study revealed that for 30% of the testing topics related to testing techniques, the proportion of job advertisements seeking expertise in these areas was higher than the proportion of universities teaching them. Conversely, for all testing levels the study revealed that the percentage of courses offering them was greater than the percentage of job advertisements seeking them. Regarding test management topics, 33% appeared less frequently in curricula than in job advertisements. Based on these findings, Hanna developed a taxonomy highlighting the topics with the largest discrepancies between university offerings and industry needs.

Two recent studies have examined software testing education in Europe. In Sweden, Barrett et al. [12] analyzed 25 universities offering computer science or related degrees in 2023. They found that 56% of these institutions offered dedicated software testing courses, with most of these courses provided at the master's level. Additionally, software testing typically accounted for only about 5% of the total degree credits.

In a broader European context, Tramontana et al. [74] analyzed the course offerings of 117 programs across Belgium, Italy, Portugal, and Spain in 2024. They reported that only 39% of universities offered a fully dedicated software testing course, which, similar to the Swedish case, was most frequently provided at the master's level.

Although a substantial body of research exists on software testing education, the current state of software testing in academic curricula still exhibits a significant mismatch with industry needs. On one hand, several secondary studies highlight active research in the field and identify ongoing challenges in teaching software testing. On the other hand, numerous studies focus on teaching practices designed to address these challenges, often evaluated with small student groups. Nonetheless, when examining the actual integration of software testing into existing curricula, it becomes evident that such integration remains limited.

These studies on the current state of software testing curricula primarily examined course offerings and the gap between academia and industry. However, they generally analyzed course descriptions rather than the detailed teaching content and materials employed by instructors. The focus was on broad topics covered, rather than on specific testing techniques, tools, and educational resources actually used in classroom practice. This limitation directly informed our Research Question 2 (RQ2) on testing topics and materials, allowing us to collect detailed data on what is actually taught and providing the instructor perspective that is often missing from curriculum-focused analyses.

3 Survey Study Design

In this study we were interested in a deeper understanding of the state of the practice of teaching software testing in academic institutions from European countries. To this aim, we conducted a survey among professors of testing courses to obtain a detailed view of the teaching approaches, topics taught, materials used, and learning and evaluation strategies.

We used the Goal-Question-Metric template [16] to define the goal of the study as follows: *Analyse software testing courses at the academic level and their characteristics for the purpose of understanding the state of the practice with respect to software testing education from the point of view of teachers in the context of higher education.*

In order to obtain a large number of participants in the survey, we decided to use an unsupervised approach, explaining the aim of our study at the beginning of the questionnaire.

3.1 Research Questions

We have formulated the following research questions to reach the goal of our study:

RQ1 What are the organizational characteristics of the courses?
Rationale: Taking into account the organisational characteristics of the courses in terms of degree level (bachelor or master), the year in which the course was offered, the number of European Credits (ECTS) and the number of students enrolled, we can analyse the information collected by categorising it and draw tailored conclusions. ECTS represents an european standard system for measuring the effort required by a student for a course that is applicable to all the selected courses.

RQ2 Which software testing topics are taught? Which testing materials are used?
Rationale: As discussed in Section 2.3, the prevalence of testing topics has been investigated in the literature by using public information from software engineering or testing courses, which usually present the topics and materials used in a generic way, leaving teachers free to choose the topics they want to teach in depth. We consider that an analysis of the topics taught and the materials used from the teachers' perspective will provide a deeper understanding of the discrepancy between theory and practice in software testing education.

RQ3 Which testing teaching practices are adopted?
Rationale: In literature (Section 2.2) we found that the main teaching practices for improving testing education are the advancement of testing topics in the first courses, the use of free, open source and real projects in the assignments, the use of automated assessment and the use of active learning approaches. However, we want to explore the prevalence of these practices and a detailed understanding of active learning practices given the characteristics of the courses.

RQ4 Which educational tools are adopted?
Rationale: Considering the need to run the software to test it and the need of interactivity that the students of the XXI century require, the knowledge of educational tools is essential. Therefore, we want to know the tools suggested by teachers to support the students practicing with tests and the ones that allow to guide the students learning path.

RQ5 Which gamification approaches are followed?
Rationale: Gamification has been shown to be effective in improving motivation and practice of complex topics such as testing. With this question we want to explore the use of gamification and the main gamified elements included in these approaches. In addition, for teachers who have not yet used gamification, we want to know if the teacher is willing to use it in some activities of the course.

RQ6 Are the teachers satisfied with the available teaching materials? What are their main desiderata about them?
Rationale: Given that teachers are already using materials to facilitate testing education, we want to know their

perceptions and opinions to understand if there is room for improvement. We believe that this information could bring light on future research to improve testing education.

3.2 Survey Design

The survey is a structured cross-sectional web-based survey. This approach helps to reach practitioners from geographically diverse locations and supports automated data collection.

To guarantee the privacy of the participants, the survey was sent for approval to the ethical committee of the university of some of the authors.

The privacy was ensured by the fact that at no point during answering the questionnaire, personal data were asked or collected (no names, addresses, etc.). The form was hosted via Microsoft Forms, as it is known to not keep track of any IP addresses of the participants either.

The questionnaire is structured in six sections. Section 1 contains questions about the course organization while Section 2 has questions about the software testing topics taught. Section 3 includes questions about the teaching practices used. Section 4 lists questions about the educational tools used. Section 5 contains questions on the usage of gamification approaches. Finally, Section 6 presents questions on teachers' opinions on the quality of their teaching. The questions of the six sections of the survey are reported in Table 1.

3.2.1 Course characteristics. The first section of the questionnaire includes questions aimed at answering RQ1 by asking information about the course organization. In particular, because of the anonymity of the responses, we did not collect the names of the instructors, the names of their courses, or the universities were they were taken, but we did collect a great deal of information about the type of course, the demographics of the students, the method of assessment, and the testing topics included in the course.

A first subset of questions aims at understanding the placement of the course: the degree level (Bachelor or Master) and the year in which it is offered, number of ECTS credits, average number of enrolled students, average age of the students, if the course is mandatory or optional, the type and number of student assessments that are done. The type of course is also identified. Here, we make a distinction between courses that are fully dedicated to software testing (ST) and courses in which only a few chapters are dedicated to software testing (NST).

3.2.2 Testing topics and materials. A second subset of questions is geared towards the structure of the course and is needed to answer RQ2. There are questions concerning the numbers of theoretical and practical lecture hours spent to software testing topics and which recommendations the teacher followed for designing the course among the ACM CS, CE, and SE Curricula Recommendations[2], the ISO-IEEE Standard 29119 – 2022 - "Software and systems engineering — Software testing"[3], and the SWEBOK - Software Engineering Body of Knowledge[4].

Subsequent questions are posed to collect more details about the specific testing topics taught, defined according to the testing terminology provided by the ISO/ IEEE Standard 29119-2022, which is summarized in Table 2. The questions ask what test design techniques, testing practices, testing levels, and testing types (according to the ISO/ IEEE Standard 29119-2022) are taught in the course, and what testing tools (e.g. Selenium, EvoSuite, Randoop, JMeter, PIT, etc.), testing frameworks/ libraries (e.g., JUnit, Mockito, Cucumber, ...) are presented. Eventually, the programming

[2]ACM Curricula Recommendations, https://www.acm.org/education/curricula-recommendations
[3]IEEE Standard 29119 - 2022, https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-2:v1:en
[4]SWEBOK - Software Engineering Body of Knowledge, https://www.computer.org/education/bodies-of-knowledge/software-engineering

Table 1. The questionnaire distributed to the teachers.

Section 1
1.1 - Is the course you teach focused on testing?
1.2 - At what level is the course taught?
1.3 - At what year(s) of the degree is the course offered?
1.4 - How many credits (ECTS) correspond to the course?
1.5 - What is the number of enrolled students in the past (or current) edition of the course?
1.6 - What is the average age of the students?
1.7 - Is the course mandatory or optional?
1.8 - How many hours of the course are devoted to theoretical lectures on software testing topics?
1.9 - How many hours of the course are devoted to practical lectures on software testing topics?
1.10 - How are the students evaluated?
1.11 - How many student evaluations do you perform?
Section 2
2.1 - Did you consider the ACM Curricula Recommendations (*) in the design of this course?
2.2 - If yes, which ACM Curricula Recommendations and which parts of them were considered?
2.3 - Did you consider the SWEBOK-Software Engineering Body of Knowledge (*) in the design of your course?
2.4 - If yes, which parts of the SWEBOK were considered?
2.5 - What test design techniques (according to the classification of Test Design Techniques provided by the ISO/ IEEE Standard 29119-2022) (*) are taught?
2.6 - What testing practices (according to the classification of Testing Practices provided by the ISO/ IEEE Standard 29119-2022) are taught?
2.7 - What testing levels (according to the classification provided by the ISO/ IEEE Standard 29119-2022) are taught in the course?
2.8 - What testing types (according to the classification provided by the ISO/ IEEE Standard 29119-2022) are taught in the course?
2.9 - What set of testing tools (e.g. Selenium, EvoSuite, Randoop, JMeter, PIT, etc.) are presented during the course?
2.10 - What set of testing frameworks/ libraries (e.g., JUnit, Mockito, Cucumber, ...) are presented during the course?
2.11 - What do you expect as students' prerequisite knowledge?
2.12 - What types of Teaching materials do you use?
2.13 - What Textbooks do you use? (authors, title, editor, year)
2.14 - What are the programming languages/ technologies of the software being tested in the course?
Section 3
3.1 - Is the TDD (Test Driven Development) practice of writing tests before writing code used in the course?
3.2 - Do you use any automated assessment of students' testing skills?
3.3 - What kind of software is tested in the classes/ homework?
3.4 - Is the Case-Based-Learning (*) practice used in the course?
3.5 - Is the Adaptive Learning Model (*) practice adopted?
3.6 - If you teach in a Programming Course, do you adopt the practice of early introducing testing?
3.7 - If yes, what are the first moments you talk about testing towards the students?
Section 4
4.1 - Do you use any of the listed types of tools/ environments for supporting students' testing learning?
4.2 - What are the main features offered by the adopted tools supporting Testing Learning?
4.3 - What test design techniques are supported by the adopted tools for Testing Learning?
4.4 - What testing practices are supported by the adopted tools for Testing Learning?
4.5 - What testing levels are addressed by the adopted tools for Testing Learning?
4.6 - What types of testing are supported by the adopted tools for Testing Learning?
4.7 - If you don't use any tool/ environment supporting Testing Learning, what type of tool/ environment would you be interested in introducing in your course? For which testing topic?
Section 5
5.1 - Do you use any Gamification Approach for testing learning?
5.2 - What Testing Activities are the objects of the Gamification Approaches?
5.3 - What Testing Techniques are the objects of the Gamification Approaches?
5.4 - Do you use any testing tool to implement your gamification approach (e.g. the Code Defenders tool (*))? What are the names of these testing tools?
5.5 - Which types of Gamification elements are adopted?
5.6 - Can you provide a brief description of how the games work? Please provide link to the games if it is possible
5.7 - If you don't use gamification, do you think that a gamification approach could be useful to help the students learning any of the testing topics presented in the course?
5.8 - If yes, which testing activities does it support?
Section 6
6.1 - Are you satisfied with the Teaching Materials currently used in the course?
6.2 - Which testing topics taught in the course (if any) would need improved Teaching Materials?
6.3 - What types of Teaching Materials would you like to be improved (such as Books, Solved Examples, Testing Exercises, ...) ?
6.4 - If you are not satisfied with some of the Teaching Material you use, do you have any suggestion about how to improve those materials?

languages/ technologies of the software being tested in the course are asked about, as well as the expected prerequisite knowledge, the types of teaching materials and textbooks that are used.

3.2.3 Testing Teaching Practices. The third part of the survey is dedicated to RQ3 and regards the testing teaching practices recommended in the literature that are adopted by the teachers. The six practices considered are:

- **Case-Based-Learning**, an interactive learning technique that includes real-world examples followed by discussions, team work, decision making tasks, brainstorming, and presentations.

Table 2. Testing approaches according to the ISO/IEC/IEEE 29119 standard on Software Testing

Test Design Technique	Testing Practice	Testing Type
Specification Based	Model-based testing	Functional testing
Equivalence Partitioning	Scripted testing	Accessibility testing
Classification tree method	Exploratory testing	Compatibility testing
Boundary value analysis	Experience-based testing	Conversion testing
Syntax testing	Manual testing	Disaster recover testing
Combinatorial testing	A/B testing	Installability testing
Decision table testing	Back-to-back testing	Interoperability testing
Cause-effect graphing	Mathematical-based testing	Localization testing
State transition testing	Fuzz testing	Maintainability testing
Scenario testing	Keyword-driven testing	Performance related testing
Use case testing	Automated testing	Portability testing
Random testing	Other	Procedure testing
Metamorphic testing		Reliability testing
Requirements-based testing		Security testing
Structure Based	**Testing Level**	Usability testing
Statement testing	Unit testing	Other
Branch testing	Integration testing	
Decision testing	System testing	
Branch condition testing	System integration testing	
Branch cond. comb. testing	Acceptance testing	
MC/DC testing	Other	
Data flow testing		
Experience Based		
Error guessing		
Other		

- **Adaptive Learning**, an approach where students are engaged in activities which aim at facilitating the acquisition of new knowledge and skills in a learner-centered manner. As an example, an adaptable learning model allows for variation in the paths to learning outcomes on the basis of the feedback on students' performance.
- **Early Testing** consists of introducing testing early on in the curriculum when the basics of programming are being taught.
- **Automated Assessment** requires using automated assessment of students' testing skills, to provide useful feedback.
- **Test-Driven Development** requires introducing the Test-Driven Development (TDD) practice in which testing is done before coding.
- **Non-Toy Software** recommends using free, open-source, industrial software, or software made by other students to be tested in classes/ homework, instead of toy examples.

3.2.4 Educational tools. The fourth section of the survey is dedicated to assessing the use of educational tools that support the teaching of software testing (RQ4). We exemplified this concept by providing the following set of categories and examples:

- Virtual Learning Environments offering information, teaching materials about testing, and social and media networking features for supporting students, like WRESTT-CyLE [38, 39];
- Tools that help students explore testing topics and practice the application of fundamental testing techniques by means of challenges with immediate feedback, like the Bug Hunt tool [31];

- Tools that can guide the students through a predefined path of steps to accomplish testing tasks;
- Tools supporting an instructional method based on machine learning.

3.2.5 Gamification Approaches. The fifth part of the questionnaire is dedicated to gathering information about the gamification approaches possibly adopted by teachers in their courses. More specifically, we asked if the teachers introduced gamified activities in their courses. If they do, we asked which testing activities are involved, which gaming tool is involved and what characteristics it has. In particular, we considered the following basic gamification elements, according to the definitions proposed by Pedreira et al.[59]:

- Awards: a particular award is given to the player on the completion of a behaviour.
- Point-based reward system: the players obtain a reward in the form of points on the completion of a certain behaviour.
- Badges: they represent certain achievements of the user.
- Levels: related to the point-based rewards; the users have a level that increases as they reach a certain number of points.
- Quests: the tasks the player must complete are presented as a quest, with additional game elements (such as a story) that makes it more attractive.
- Voting: players can vote on another player's behaviour. The votes themselves represent the rewards obtained by each player.
- Ranking: a ranking with the top players is presented to all players to increase competitiveness. The position in the ranking can be defined by points, levels, or number of votes, for example.
- Betting: users bet on a certain event, such as an estimation, for example. The winner of the bet receives some reward in exchange.

In case the teacher did not use any gamification element, we also asked if the introduction of such activities would be considered to be potentially useful in teaching testing.

3.2.6 Teachers' opinions. The last section of the survey is devoted to collecting teachers' opinions and comments regarding the satisfaction for the existing testing teaching materials, their educational support, and their wishes/suggestions about the future development of further support and new tools. These information are gathered by means of questions with free text answers.

3.3 Teachers' Selection

After obtaining ethical approval for the questionnaire, the authors collected a list of names of teachers who might be interested in participating in the survey.

Two criteria were considered for including teachers in this list: (1) the professor is currently teaching an academic course including software testing topics (i.e. programming courses or software engineering courses) or completely devoted to software testing and (2) the course is currently offered in a western European country. We used these criteria to select a sample of qualified teachers of software testing courses who belonged to the geographic area we decided to focus on. To implement these criteria, we based our search for participants on the web sites of the courses offered by academic institutions, as also done in other works [74].

The search for teachers started from a list of universities offering Computer Science courses categorized per country. A list of 605 universities from western European countries involving Computer Science researchers is provided by the Scimago Institution Ranking[5].

The authors randomly selected universities from this list from different countries (Austria, Belgium, Denmark, Finland, Germany, Iceland, Italy, Malta, Portugal, Spain, Switzerland) and analyzed the course offerings of each of them, focusing on Computer Science and Computer Engineering Bachelor and Master degrees. In this way they collected a list of 143 teachers who are involved in teaching at least one academic course which includes software testing topics (as it can be deduced by the syllabi of the courses).

3.4 Survey Distribution

Invitations to participate in the survey study were exclusively sent by e-mail over the period from May 2023 to February 2024 to the identified teachers. The survey was not distributed via the broadcast channels, to ensure that responses were collected only from clearly qualified teachers who satisfied the defined inclusion criteria. We collected 49 valid answers of teachers who completely answered the proposed questions with reference to the course they taught that included testing topics.

3.5 Data collection and analysis procedures

The survey was conducted using Microsoft Forms, and the responses were exported in tabular format. To ensure data quality, two of the authors independently carried out an initial filtering process to verify compliance with the inclusion criteria. This step helped guarantee that only responses relevant to the study's scope were retained; for instance, two responses were discarded because the teachers reported that their courses did not cover software testing topics.

The same two authors also addressed inconsistencies across responses and coded the open-ended questions. This double-checking process enhanced reliability by reducing the risk of misinterpretation and ensuring that subjective judgments (e.g., whether an item should be considered a tool or a framework) were consistently applied. In cases where responses were incomplete or clearly inconsistent with the intended question, those entries were excluded to avoid introducing noise into the analysis.

Finally, all authors collectively reflected on the interpretation of the data and the presentation of the results. This collaborative step contributed to construct validity, as it combined multiple perspectives to minimize bias in the analysis.

The anonymized survey dataset is publicly available online[6].

4 Results

4.1 RQ1 : What are the organizational characteristics of the courses?

The respondents were teachers of 32 bachelor's degree courses and 17 master's degree courses. Teachers were asked to indicate whether their courses were entirely dedicated to software testing topics (hereafter referred to as Software Testing Courses - ST) or whether they were not solely focused on software testing but included some testing topics (hereafter referred to as NST).

The teachers taught 22 ST courses and 27 NST courses. The distribution of these two kinds of courses over the academic years is shown in Figure 1. The majority of the ST courses are offered in a Master degree (14 out of 22), while

[5]Scimago Institution Ranking limited to Western Europe and Computer Science, available at: https://www.scimagoir.com/rankings.php?sector=Higher+educ.&country=Western+Europe&area=1700
[6]Anoymized results from the survey, https://anonymous.4open.science/r/TeacherSurveyStudy-2B1D/

the majority of the NST courses are offered in a bachelor degree (24 out of 27). Note that there is a course taken at the 4th year of Bachelor degree. Bachelor degrees usually consists of three academic years but there are some countries in which fourth year Bachelor courses are possible (e.g. in Spain).

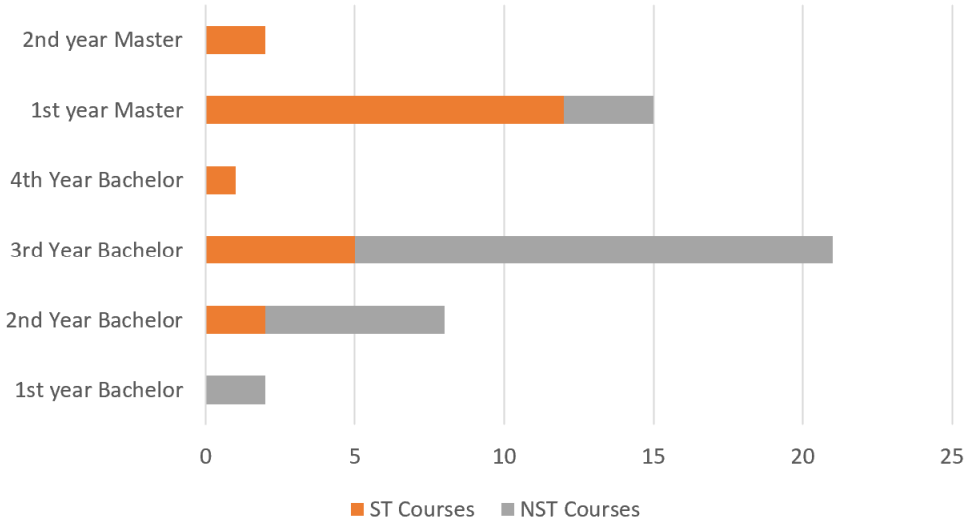

Fig. 1. Distribution of Courses per Year and Degree type

Concerning the size of the courses, we asked for the number of European Credits (ECTS) associated with each course. In our sample, ST courses on average had 5.5 ECTS, whereas NST on average had 6.9 ECTS. More specifically, ST courses ranged from 2 to 7.5 ECTS and most frequently consisted of 6 ECTS (in 8 cases out of 22) whereas NST courses vary between 3 and 12 ECTS (most frequently 6 and 9 ECTS).

Both ST and NST courses have similar distributions of the time allocated to theoretical lectures and practical activities: in ST courses there are, on average, 18.5 hours devoted on theoretical lectures and 18.3 hours on practical activities. In NST there are on average 13.3 hours allocated to testing of which 6.1 allocated to theoretical lectures and 7.2 hours to practical activities.

The number of students participating in these courses is quite different between the two categories: there are on average 63 students per year for each ST course and 137 students per year in the NST courses. This difference may be explained by considering that ST courses are generally taught at master's degree level, where the total number of students is generally lower. In addition, we observe that only 15 out of 22 analysed ST courses are mandatory, whereas the NST course are predominantly mandatory (24 out of 27). This can again be explained by the fact that ST courses are rather taught at the master level, in which more elective courses are present.

Table 3 summarises the aggregated data of the 49 surveyed courses.

4.1.1 Assessment Methods. As regards the assessment methods, the participants of the survey adopt a range of different evaluation methods. More specifically, teachers of the ST courses on average mentioned the adoption of 2.8 different assessment methods, with the highest prevalence towards Practical Projects (in 77% of courses). Moreover, in most of the courses (15 out of 22) more than two different assessment methods were considered. In the NST courses we

Table 3. Statistics about Organizational Characteristics of the Analysed Courses

		ST	NST
Courses	#	22	27
Bachelor	#	8	24
Master	#	14	3
Mandatory	#	15	24
Optional	#	7	3
ECTS	avg.	5,5	6,9
Theoretical lecture	avg. hours	18,5	6,1
Practical hours	avg.	18,3	7,2
Enrolled Students	avg. #	63	137

Table 4. Number of different assessment methods adopted by teachers and number of student evaluations

Number of different assessment methods				
	ST		NST	
One	4	18%	9	41%
Two	3	14%	11	50%
Three or more	15	68%	7	32%

Number of student evaluations				
	ST		NST	
One	5	23%	13	59%
Two	1	5%	6	27%
Three or more	16	73%	8	36%

observed that on average only 2 different assessment methods were used (20 out of 27 courses adopted just one or two assessment methods), with the highest prevalence towards Practical Projects and Exercises (both used in 56% of courses). These data seem to show that practical activities (e.g. projects, exercises, homeworks) are considered mostly in ST courses, whereas oral exams are relatively rare. Namely, only 10 teachers out of 49 consider the latter assessment method.

The teachers of ST courses declared also that the assessment was predominantly distributed in three or more tasks during and after the course (in 16 out of 22 courses), whereas in most of the NST courses the evaluation was distributed in just a final task (in 13 out of 27 NST courses) or in two tasks (in 6 courses). Only the remaining 8 courses (36%) adopted an evaluation in three or more steps.

As explained by some teachers, in many NST courses the large number of students entailed the adoption of a traditional final evaluation of students after the course, whereas in ST courses many teachers succeeded in managing three or more assignments during and after the course.

Table 4 summarizes the number of different assessment methods adopted by teachers and the number of evaluations each student has during and after the course while Figure 2 shows the distribution of the different assessment methods.

These survey results confirm the potential for improving software testing education across different degree programs and, in particular, at the bachelor level. As observed in the results, considering that most dedicated software testing courses are offered at the master's level, there can be some risks that when bachelor graduates want to join industry without pursing a master's degree, they may lack sufficient testing knowledge. Also, more courses at the master's level

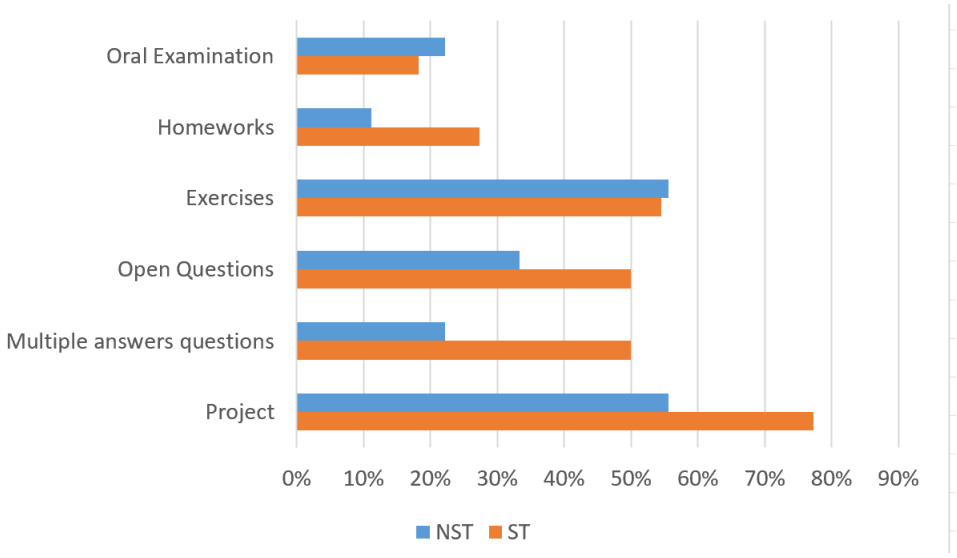

Fig. 2. Frequency of Adoption (in percentage) of Different Assessment Methods

including ST ones can be offered as non-mandatory and elective courses. Therefore, these factors can contribute to the gap between academic education in software testing and industry needs and expectations.

The results also highlight the applicability of various assessment methods in ST courses with distribution in more tasks throughout the course (rather than just a final task and exam). This also implies that there can be a higher possibility of modularization in designing the content and delivery of ST courses, for instance, defining separate teaching modules and assessments for different testing techniques and practices, such as mutation-based testing, combinatorial testing, and so on. This topic can be interesting to investigate further and evaluate more carefully in future studies.

4.2 RQ2 : Which software testing topics are taught? Which testing materials are used?

4.2.1 Course Design. In order to understand in more detail how teachers have designed their courses, we asked if they followed the recommendations and taxonomies of some standard sources, such as ACM or SWEBOK Curricula recommendations.

We observed that ACM and SWEBOK have both a limited diffusion among teachers: only 13 out of 49 teachers considered at least one of them for course design (27%) and only 5 of them (10%) considered both. More specifically, ACM Curricula were considered by only 2 teachers when designing ST courses (9%) and by 6 teachers designing NST courses (22%), whereas SWEBOK was considered by 5 teachers for ST courses (23%) and 5 teachers for NST courses (19%).

The collected answers appear to indicate that ACM Curricula is too generic about testing topics for a ST course, while it could be more useful for designing a NST course. SE2014 is the most cited source, followed by CSSE376. Analogously, Chapter 4 of SWEBOK was cited as one of the resources used by some ST teachers. However, several teachers mentions that this was only used for inspiration or framing of the course. One reason for this could be that it might not be easy to find the desired information in these resources. For example, ACM provides several Curricula Recommendations spread over distinct guidelines: *software verification & validation (VAV)* is a software engineering

education knowledge area within SE2014 (Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering), *SE-Validation: Software Verification and Validation* is a knowledge unit inside the *Software Engineering* knowledge area of CS2023 (Computer Science Curricula 2023), *Software testing and quality* is a knowledge unit inside the *Software Design* knowledge area of CE2016 (Computer Engineering Curricula 2016), and *Testing* is a knowledge area of GSwE2019 (Graduate Software Engineering 2009). In addition, software testing topics may be related to more than one knowledge area. For instance, in CE2016, we also find *System integration, testing and validation* within *Systems and Project Engineering*. On the other hand, SWEBOK has a full chapter for *Software Testing*. In addition, while there are topics covered by both ACM and SWEBOK, such as unit testing, there are others that are explicitly mentioned only in SWEBOK, such as pairwise testing. A more consistent and focused recommendation across resources for software testing might motivate teachers to follow it more often.

In addition, as mentioned above, some teachers considered only SWEBOK and others only considered the ACM Curricula. Although we do not have enough information to conclude with certainty why one was favored over the other, one reason could be related to the focus of each teacher's course, as SWEBOK is generally seen as more oriented towards industry whilst ACM Curricula is seen as more academically focused.

4.2.2 Testing Topics. We analyzed the answers related to the test design techniques taught in the context of the courses. Figure 3 shows the percentage of courses that adopt each of the techniques cited in the ISO/IEC/IEEE 29119 standard.

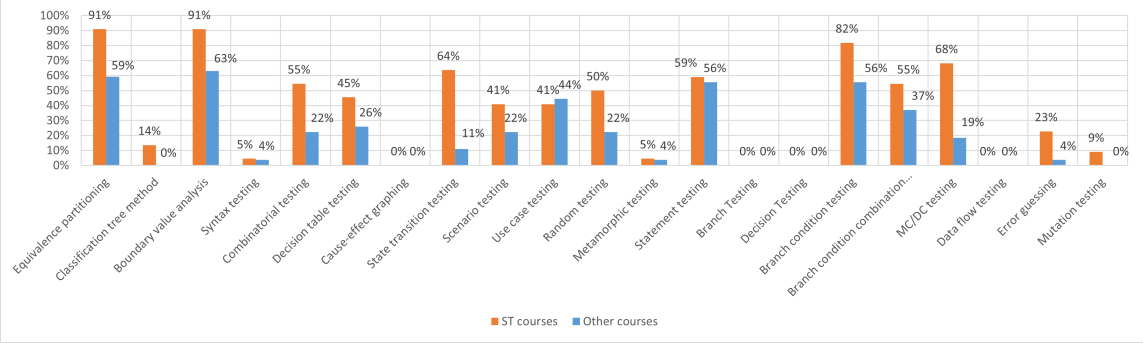

Fig. 3. Test Design Techniques Taught in ST and NST Courses

On average, there are approximately 8.3 different testing topics for each ST course and 5 different testing topics for NST courses. The most commonly included topics in ST courses belong to Structure-based techniques (Branch condition testing, MC/DC testing, Decision Testing, Branch Testing, Branch condition combination testing and Statement testing, in order of decreasing frequency) and to a lesser extent Specification-based techniques (Boundary value analysis, Equivalence partitioning, State transition testing), while Experience-based techniques are not frequently covered in these courses.

For the NST courses, the six most commonly taught testing techniques belong mostly to Specification-based techniques (Requirements-based testing, Boundary value analysis, Equivalence partitioning, Use case testing) and to the lowest extent to Structure-based techniques (Branch Testing, Branch condition testing). Also in this case Experience-based techniques are not in the list of the most frequently included techniques.

NST courses are generally taught earlier in the curriculum compared to ST courses and specification-based techniques are considered to be more basic techniques compared to structure-based ones, which are mostly present in ST courses.

The least prevalent techniques (present in less than 15% of the considered courses) are Syntax testing, Cause-effect graphing, Error guessing, Metamorphic testing and Data flow testing for the ST courses and Cause-effect graphing, State transition testing, Error guessing, Classification tree method, Metamorphic testing, Syntax testing, Data flow testing and Mutation testing for the NST courses.

Figure 4 shows how often the testing practices proposed by the ISO/ IEEE Standard 29119-2022 are included in the surveyed courses. We observe that in software testing course the most popular practices are Automated testing (in 86%

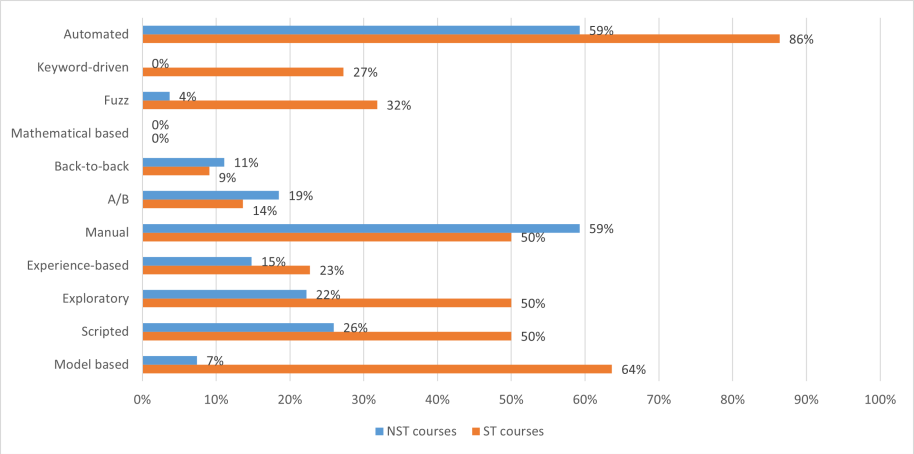

Fig. 4. Testing Practices Taught in ST and NST Courses

of courses) and Model based testing (64%), followed by Manual testing, Scripted testing and Exploratory testing (all declared by 50% of the teachers of ST courses). On the other hand, Automated testing and Manual testing are both taught in 59% of NST courses whereas all the other practices are taught in less than 30% of the NST courses.

For the testing levels (see Figure 5), we observe that there are no major differences between ST and NST courses: Unit testing is taught almost in each course and System testing is the only testing level for which there is a significant difference between software testing and other courses: it is taught in 77% of ST courses and only in 52% of NST courses. This difference may be explained by the greater amount of time that can be devoted to teaching more testing levels in ST courses than in NST courses.

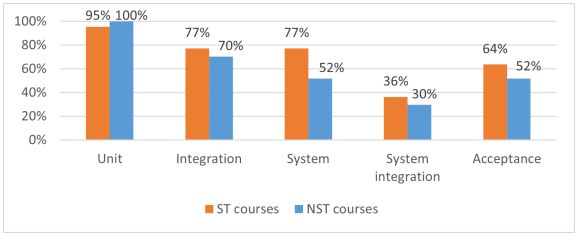

Fig. 5. Testing Levels Taught in ST and NST Courses

The last question about testing topics considered different testing types. We observe that in all the courses there is little space for non-functional testing whereas functional testing is almost always taught (in 95% of ST courses and 96%

of NST courses). On average ST courses cover one other type of testing while NST courses on average cover another 0.6 types. Performance testing is the most frequently taught testing type in ST courses (36% of courses), followed by usability testing (23%). For NST courses the most taught testing type is security testing (in 15% of courses), followed by performance testing (in 11%).

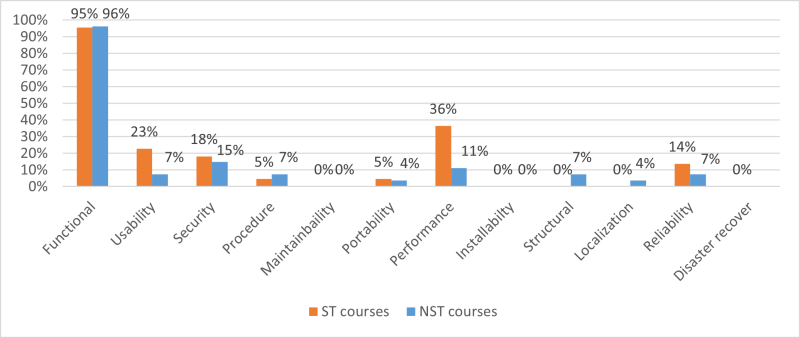

Fig. 6. Testing Types Taught in ST and NST Courses

This scarce diffusion of non-functional testing types may be due both to the limited attention in university courses for quality related aspects and to the fact that some of them are considered more related to other types of courses (e.g. security testing is often taught in security courses and usability testing is taught in Human-Computer Interaction courses as we have observed by the analysis of the design of the degrees).

4.2.3 Students' prerequisite knowledge. The prerequisite knowledge expected from students when enrolling in an ST course were quite limited: 16 ST courses (73%) require only basic programming skills, whereas 5 courses (23%) require advanced software engineering skills and one course requires no prerequisite knowledge.

The prerequisite knowledge required for NST courses is similar: 19 of them (70%) require basic programming skills, while there are 2 courses (7%) that require basics software engineering skills and 5 courses (18%) that require advanced software engineering skills. Only one course (4%) had no prerequisite.

This data suggests that most Software Testing topics are not considered advanced Software Engineering topics and, therefore, can be taught right after basic programming courses.

4.2.4 Testing Materials and tools. For the software testing materials, we asked different questions to know, respectively, which tools, frameworks and textbooks are used by teachers to support teaching testing.

Most of the ST courses use slides (95%), whereas some NST courses avoided them (they are used in 81% of those courses). Other materials (videos, scientific papers) are equally used in a minority (about 18%) of courses. Practical exercises are used in the majority of courses (64% of ST courses and 59% of NST courses). Finally, it is interesting to observe a limited use of books: only 41% of teachers of ST courses and 15% of teachers of NST courses use books to teach testing.

We also observe a very large and unexpected fragmentation in books: we found 27 different testing books. The most popular testing books are the ones of Young and Pezzè ("Software Testing and Analysis: Process, Principles and Techniques" [82]) and of Mauricio Aniche ("Effective Software Testing: A Developer's Guide"[5]) each one adopted by 6 teachers (12%), while the one of Ammann and Offutt ("Introduction to Software Testing" [4]) is used in 3 courses

Table 5. List of Books Adopted by Surveyed Teachers with their Occurrences

Authors	Title	Occ.
Aniche	Effective Software Testing: A Developer's Guide	6
Pezzè, Young	Software Testing and Analysis: Process, Principles and Techniques	6
Beck	Test Driven Development. By Example	5
Amman, Offutt	Introduction to software testing	3
Burnstein	Practical software testing: a process-oriented approach	2
Crispin, Gregory	Agile Testing: A Practical Guide for Testers and Agile Teams	2
Jorgensen	Software Testing A Craftsman's Approach	2
Meszaros	xUnit Test Patterns: Refactoring Test Code	2
Bolaños, Sierra, Alarcón	Pruebas de Software y JUnit	1
Copeland	A Practitioner's Guide to Software Test Design	1
Forgács, Kovács	Practical Test Design: Selection of traditional and automated test design technique	1
Fowler	Test Driven Development	1
Fraser. Rojas	Software Testing	1
Graham, Black, van Veenendaal	Foundations of Software Testing: ISTQB Certification	1
Gregory	More agile testing: learning journeys for the whole team	1
Gulati, Sharm	Java Unit Testing with JUnit 5 : Test Driven Development with JUnit 5	1
Kaczanowski	Good Tests	1
Kan	Metrics and models in software quality engineering	1
Kaner	Testing computer software	1
Khorikov	Unit Testing	1
Mathur	Foundations of Software Testing	1
Molyneaux	The art of application performance testing: from strategy to tools	1
Rainsberger	JUnit recipes: practical methods for programmer testing	1
Spillner, Linz	Software testing foundations - a study guide for the certified tester exam : foundation level, ISTQB compliant	1
Utting	Practical Model-Based Testing	1
Van Veenendaal, Graham, Black	Foundations of Software Testing: ISTQB Certification	1
Vos, van Vugt-Hage	Software Testing	1

(6%). In addition, the book of Kent Beck specialized on TDD ("Test Driven Development: By Example" [13]) is used in 5 courses (10%).

Table 5 reports the list of books declared by the survey respondents, with the number of occurrences (of 49 courses) of each one. We removed books that are not directly related to software testing from this list. For example, several software engineering teachers declared that they use the Sommerville book on Software Engineering [67], which includes some chapters dedicated to software testing but cannot be defined as a software testing textbook. Analogously, we removed the programming books.

Concerning the testing tools, the most popular one is Selenium[7], that is used for the generation and the execution of End-to-End test cases for Web applications: it is adopted in 20 out of 49 courses (41%). Other tools are often used to teach specific topics. For mutant generation the most popular tool is PIT[8] (in 8 courses, 16%) which is able to automatically generate mutants for Java applications. For unit testing the most popular tool is Evosuite[9] (in 5 courses, 10%), which automatically generates JUnit test cases with a search-based approach. For performance testing the most used tool is JMeter[10] (in 7 courses, 14%) which measures the performance of running applications. For white box testing JaCoCo[11] is the most used library (in 3 courses, 6%). The left side of Table 6 reports the complete list of tools used in courses taken by the surveyed teachers, with the number of courses in which each tool is adopted. The most popular tools are all free and/or open source tools. It is interesting to see that 14 out of 49 teachers (29%) declared that they do not use any tool in their courses.

[7]Selenium, https://www.selenium.dev/
[8]PIT, https://pitest.org/
[9]Evosuite, https://www.evosuite.org/
[10]JMeter, https://jmeter.apache.org/
[11]JaCoCo,https://www.eclemma.org/jacoco/

Table 6. List of Tools and Frameworks Used by Surveyed Teachers with their Occurrences

Tool name	Occ.	Framework name	Occ.
Selenium	20	JUnit	38
jMeter	7	Mockito	12
PIT	7	Xunit	4
Evosuite	5	Cucumber	2
JaCoCo	2	jMock	2
NuSMV	2	Nunit	2
Randoop	2	AssertJ	1
SonarQube	2	DBUnit	1
Asmeta	1	Jbehave	1
CodeCover	1	jqwik	1
CodeDefenders	1	PHPUnit	1
EclEmma	1		
GraphWalker	1		
Jenkins	1		
jQwik	1		
NightWatch	1		
OpenJML	1		
Pex	1		
Pitest	1		
QFTest	1		
TestCompass	1		
TESTONA	1		
Yakindu	1		

The most used framework/library are the ones of the XUnit family (42 out of 49 courses, 86%), that are able to support the development of test cases at different levels beyond unit testing (mostly, JUnit is used). For unit and integration testing, the most used framework supporting the generation of mock objects is Mockito (12 out of 49 courses, 24%), while JMock is used by two other teachers (4%). The Cucumber framework is used by two distinct teachers for supporting acceptance testing (4%). Other libraries and frameworks are used in just one course. It is interesting to see that Mockito is mostly used in ST courses (10 out of 12), whereas the other frameworks are used both in ST and in NST courses. The right side of Table 6 shows the diffusion of the frameworks into the courses described by the surveyed teachers.

The most popular language used to teach test case design and implementation is Java, that is adopted in 82% courses (40 out of 49), while Python is used by 11 teachers. Other languages are used less, such as C# (in 5 courses, 10%), Javascript (in 4 courses, 8%) and C++ (in 3 courses, 6%). Of course, there are some courses in which the teacher leaves the students free to use their preferred language for testing.

While Java may not be the most popular programming language[12] [18], neither the most used programming language among developers worldwide[13], it is prevalent in academia. Java is based on object-oriented programming (OOP) principles, avoiding complexities such as pointers, Goto statements, and multiple inheritance, which makes it simpler and a good candidate for teaching students to program and teach OOP concepts. These may be reasons for its adoption in academia. In addition, previous studies have shown that Java is the programming language most used in academia for the first programming course (CS1) and that it is, by far, the most used for a second programming course (CS2) [64]. It has also been observed that the majority of institutions chose the same programming language for the first and second course, which can also justify the use of Java in later courses (ST and NST). Another reason for the adoption of

[12] https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language-learn
[13] https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

Java is that the most popular frameworks for testing was natively created for Java (e.g. JUnit) and for this language there is the wider support in terms of freely available supporting tools and frameworks.

4.3 RQ3 : What are the practices adopted for the teaching of testing?

The analysis of the literature revealed some specific practices for teaching testing, the diffusion of which we want to investigate. Figure 7 shows an overview of these practices and of their diffusion in software testing courses and in the other courses including testing topics.

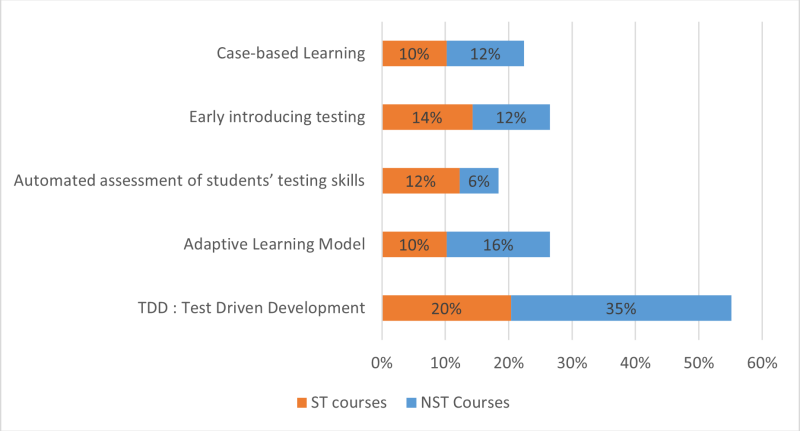

Fig. 7. Testing Testing Practices in ST and NST Courses

4.3.1 Test-Driven Development. Firstly, Test-Driven Development (TDD) is the most frequently adopted software testing teaching practice for both the ST (45%) and NST (63%) courses. Prior research has been able to find a positive correlation between students' adherence to TDD and the quality of their code and thoroughness of their testing [49, 51]. Furthermore, several training methods have also been proposed and evaluated to ensure a better integration of TDD in the curriculum [21, 68]. In fact, Scatalon et al. [62] had conducted a literature review on research towards teaching practices of software testing in education. Herein, TDD was found in 61.5% of the identified papers. The prevalence of TDD being applied in ST and NST courses of 55% (27/49) as found in this paper is thus similar to the prevalance of TDD research in the relevant literature.

4.3.2 Adaptive Learning Model. Secondly, an adaptive learning model is applied in 23% of ST and 30% of NST courses. However, 8 of the 13 teachers who have reported the usage of an Adaptive Learning Model declare that they only use it in an informal way. The integration of an adaptive learning model with software testing seems to be less elaborately researched than TDD. In fact, we were only able to find one paper with a specific focus on software testing. This is the paper by Agarwal et al. [1] who propose an adaptive learning module for teaching software testing. The paper found that the adaptive learning module allowed students to proceed through the learning material at their own pace. However, no evaluation on the effect of students' achievement on the learning outcomes of a software testing course was done. Several other papers do discuss adaptive learning in the more general context of computing education. However, only one paper contained an empirical validation towards students: Troussas et al. [75] found that the integration of an adaptive learning model within a programming course was beneficial towards students' achievement on the courses'

learning objectives. Furthermore, a scoping review on adaptive learning systems in computer science education [11] found that within computer science computer programming was the most researched topic. However, the identified papers all concerned an evaluation on the underlying adaptive learning technology (such as the recommender system) and do not consider student outcomes. A similar thing can be said about the papers identified in Jamal et al.'s [48] systematic mapping review. So, despite the relatively high prevalence of adaptive learning in both ST (23%; 5/22) and NST (30%; 8/27) courses that we were able to identify in this paper, research on the integration of adaptive learning with software testing seems to be lacking behind.

4.3.3 Automated assessment of students' testing skills. Thirdly, the automated assessment of students' testing skills is the least identified teaching practice in our research (27% of ST courses and 11% of NST courses). Nonetheless, there has been quite some ongoing research on this teaching practice. Scatalon et al. [62] has found eight papers that researched an automated assessment approach. Furthermore, another literature review by Paiva et al. [57] on the integration of automated assessment in computer science education was able to identify several research papers specifically on automated assessments in software testing. Four papers researched tools which allow for automated output comparison were identified, as well as six papers that discuss an automated unit testing assessment tool. Moreover, Garousi et al. [43] has also identified ten papers researching a specific web-based system used for the automatic grading of programming assignments. In this paper we have thus identified a relatively poor integration of these automated assessment tools for both the ST (27%; 6/22) and NST (11%; 3/27) courses.

4.3.4 Case-based learning. Fourthly, case-based learning has been identified in 11/49 (22%) of the courses (23% of ST courses and 22% of NST courses), however, previous research has mostly focused on case-based learning within the more general context of software engineering. For example, Garg et al. have explored the effectiveness of a case-based learning environment within one course [41], which was later expanded to the context of four software engineering courses [40]. Another example comes from Richardson and Delaney [60] who have analyzed the strengths and weaknesses of using problem-based learning. Based on their results they summon several recommendations on how such teaching method could be used in future courses. The prevalence of case-based learning in software testing is less as is evidenced by the fact that Garousi et al.'s [43] systematic literature mapping on software testing education only identified one paper concerning case-based learning. This is the paper by Tiwari et al. [72] in which a case-based learning approach was adopted for undergraduate students. The empirical analysis of their results shows that students performed better in five identified objectives of case-based learning. So, compared to the prevalence of adaptive learning in both ST (23%; 5/22) and NST (22%; 6/27) courses that we were able to identify for case-based learning in this paper, research on the integration of case-based learning with software testing seems to not have been widely explored yet.

4.3.5 Early introduction to testing. Introducing testing early on has been reported to be used in 32% of ST courses and 22% of NST courses. In some cases only an introduction to software testing was provided at the beginning of the NST course, while other teachers introduce the concepts of program verification and program correctness early in order to introduce test-first practices. In addition, 7 teachers of ST courses also agreed with this idea. They often have ST courses for students for whom testing was introduced early in the programming and software engineering courses. Some of them declared, too, their methodologies of introducing practical testing very early in their ST courses. For 6 out of the 27 NST courses the teacher said that they introduce testing in the introduction of the course. In one course a test-first style is implemented in which tests are used to develop their first implementation assignment. One teacher explained that their students received automated functional acceptance testing for several of their programming courses (such

as during their Object-Oriented Programming course and Software Engineering course). Similarly, for 5 out of the 22 ST courses teachers said that their students have been introduced to testing in a more basic programming course which they take prior to their ST course. One of those teachers said that he teaches his students in the first lecture that tests are the contract of what code needs to do and form a safety net when there is the need to refactor. Another teacher said that they start by teaching about delivery pipelines and that this requires the introduction of basic unit testing principles. This is followed by a lecture on testing practices. Prior research has found evidence that this teaching approach can be beneficial. Namely, Marello et al. [55] researched the effect of introducing testing earlier and placing a larger emphasize on it. They found the results to be qualitatively beneficial for students. However, this is the only research identified by Garousi et al.'s [43] systematic literature mapping on software testing education that evaluates an early introduction to testing. Moreover, Timoney et al. [70] also warn that testing is difficult for students to understand without any programming experience. Nonetheless, 13 out of 49 ST and NST courses we identified (27%) make use of an early introduction to testing. The research on this teaching approach seem to be lacking behind the integration of courses that have already adopted it, definitely considering the lack of research on the quantitative benefits for students.

4.3.6 Type of Software being Tested. Finally, we asked about the nature of the examples that are used to support the teaching of software testing. In most cases, toy examples are used (29 out of 49 courses, 59%), while another common choice consists of asking students to test programs that they have autonomously developed (in 25 courses, 51%, more often in NST courses). Free or open-source software systems are considered as case studies in 11 ST courses and 2 NST courses. Finally, there is one course in which software is specifically written and used as the object of the testing activities. This suggests that most courses do not consider using real examples used in industry.

This may indicate the difficulty in adapting free or open-source software to the classroom context, which may be due to the difficulty in configuring/installing the required software. In addition, class time is limited. Also, in an academic environment the teacher must consider that students are not all at the same pace and such "real-world" free or open-source software may be too complicated to illustrate ST concepts in simple enough terms.

4.4 RQ4 : Which Educational Tools are adopted?

The purpose of Section 4 of the survey was to investigate the extent to which instructors employ educational tools to support the teaching and learning of software testing, as well as to analyze the characteristics of the tools adopted.

Table 7 summarizes the characteristics of the educational tools adopted, reporting the number of responses collected for each of them.

Educational tools were used in only 10 ST courses. In addition, in three NST courses, other types of tools are adopted, although their categories have not been specified by the corresponding respondents.

Specifically, five teachers reported providing tools that help students explore testing topics, while another five indicated using tools that guide students in test design. Virtual Learning Environments were adopted in only three cases, and a single course employed a tool supporting learning through Machine Learning. Notably, four teachers reported adopting tools from two different categories within the same course.

As regards the topics covered by these tools, 7 of them are related to test case specification, 7 to test case implementation, another 6 to test case execution results evaluation and only 3 to the learning of general testing concepts. In 9 cases the adopted tools cover more than one of these topics.

The set of testing techniques supported by the teaching tools is quite diverse, with many tools supporting multiple techniques. The most frequently supported techniques are specification-based testing (e.g., equivalence partitioning

Table 7. Characteristics of the Educational Tools Adopted by Survey Participants

Tool Typology	#
Tools that help students explore testing topics	5
Tools that guide the students accomplish testing tasks	5
Virtual Learning Environments	3
Tools that support learning through Machine Learning	1
Supported Testing Topics	#
Test Case Specification	7
Test Case Implementation	7
Test Case Execution and Results Evaluation	7
Testing Concept Learning	3
Testing Levels	#
Unit	15
System	9
Integration	6
Acceptance	3
System Integration	2
Testing practices	#
Automated Testing	8
Model-based	4
Exploratory	3
Manual	3
Scripted	2
Experience-based	2
Back-to-back	1
Fuzz	1
Keyword-driven	1

Addressed Testing Techniques	#
Branch Testing	6
Decision Testing	6
Branch Condition Testing	6
Equivalence Partitioning	5
Boundary Value Analysis	5
Random Testing	5
Statement Testing	5
Use Case Testing	4
Branch Condition Combination Testing	4
State Transition Testing	3
MC/DC Testing	3
Classification Tree Method	2
Combinatorial Testing	2
Cause-effect Graphing	2
Scenario Testing	2
Requirements-based Testing	2
Syntax Testing	1
Mutation Testing	1

and boundary value analysis), supported by five tools; structural-based testing (e.g., branch testing, decision testing, branch condition testing, branch condition combination testing, and MC/DC testing), supported by six tools; random testing, supported by five tools; and state transition testing, supported by three tools.

The tools were mainly aimed at teaching unit level (in 12 courses) and system level (7 courses) testing, whereas in only 5 courses they were used for integration testing. Only 3 courses used a tool for acceptance testing and 2 courses use an educational tool for system integration testing. In nine courses, these tools supported more than one testing level.

From the perspective of testing practices supported by educational tools, automated testing is covered in eight courses, model-based testing in four courses, and exploratory testing in four courses, while other practices are only sporadically supported. Additionally, five teachers reported that the tools they adopted support more than one testing practice.

Educational tools are most frequently used in the context of functional testing instruction, appearing in 11 courses. In contrast, only two courses employed tools for performance testing, while static testing, disaster/recovery testing, and interoperability testing were each supported by educational tools in just a single course.

Our conclusions are that the use of educational tools is limited, being adopted by only a minority of ST courses (10 out of 22). The adopted tools are diverse and are generally developed and provided by individual course instructors,

Table 8. Characteristics of the Gamification Approaches Adopted by Survey Participants

Testing Activities	#
Test Case Design	5
Test Case Implementation	5
Test Case Results Evaluation	3
Test Case Execution	1
Test Code Review	1
Adopted Testing tools	#
Code Defenders	3
SQLTest & GoRace	1
No tools	3
Gamification Elements	#
Point-based Reward System	2
Awards	1
Badges	1
Ranking	2
None	3

Testing Techniques	#
Equivalence Partitioning	3
Boundary Value Analysis	2
Mutation Testing	2
Exploratory Testing	1
Use case testing	1
Classification Tree Method	1
Decision Testing	1
Branch Condition Testing	1
Branch Condition Combination Testing	1

allowing them to specifically organize the practical activities for their courses. As a result, there is no widespread use of general-purpose tools and supporting materials.

This situation is likely due to the limited availability of freely accessible and continuously maintained tools and materials. Rapid technological advancements render textbooks and the interactive tools they include quickly obsolete. It also reflects the fragmentation in the adoption of software testing textbooks, as discussed in Subsection4.2.4. The resources developed by teachers are generally not available to the community and their evolution is carried out only in the context of specific research projects.

We also collected additional information from 13 out of 49 teachers regarding the characteristics of the tools and environments they would be interested in introducing into their courses. Teachers generally expressed interest in collections of testing exercises that could be automatically collected from students and that support multiple testing levels, including exercises on real systems with known bugs. Additionally, one teacher requested a tool that supports traceability from requirements to testing.

4.5 RQ5 : Which Gamification approaches are followed?

The fifth part of the survey collected data on the diffusion of gamification approaches in the context of software testing education. Overall, gamification was adopted in only 7 courses (14%), including 4 software testing (ST) courses and 3 non software testing (NST) courses. In other words, it is noteworthy that in 42 out of 49 courses (86%) teachers declared not using any gamification approaches. When gamification approaches were used, they were applied to various activities: test case design (5 courses), test case implementation (4 courses), test case results evaluation (3 courses), test case execution (1 course), and test code review (1 course).

Table 8 summarizes the characteristics of the gamification approaches adopted by the teachers involved in the survey, reporting the number of responses collected for each of them.

Given the limited number of identified gamification approaches, these can be described in detail. Only in 4 cases an existing gamified tool was used, and in the remaining 3 cases, the gamification approaches did not involve the use of specific tools, but they are manually directed and managed by the teacher.

Regarding the four courses that adopt a gamified tool, three of them used the Code Defenders tool[14], whereas in the other case the combination of the SQLTest and the GoRace tools[15] was used. In the three courses adopting Code Defenders, the students are divided in two teams: in two courses, one team played first as attacker (that mutate source code) and the other team as defenders (that write JUnit test cases to kill the mutants created by the attackers) [35]. In the second round the roles were switched between the teams [37]. In another course, students instead played both as attacker and as defenders on open games [36]. In the fourth course, the combination of SQLTest and GoRace tools is used to support a complex gamification environment where a sequence of tasks involving different testing activities were asked to be solved by students.

Regarding the three cases that use gamification approaches without the support of a specific tool, in two courses gamification mechanisms are applied by introducing, for example, point-based reward systems, awards, or badges to incentive students in competitive activities based on test case design. For example, in one course a philosophical game was proposed, which did not involve the writing of executable test cases ("*We have two black balls and two coins that look the same but have differences. Students need to find the differences by measuring, weight, properties, etc.*"). In the third course, the gamification of reviewing unit test code for readability issues and suggested improvements was proposed.

Regarding the gamification elements that they adopted, there were only four answers showing the use of Point-base rewards (2), Ranking (2), Awards and Badges (1) as gamification elements. As shown by the low number of responses and the low number of gamification elements used, we believe that there is a lack of knowledge about the impact that gamification elements can have on learning.

Moreover, we asked to all teachers if they think that gamification could be a valid support for the future improvement of their software testing courses. 31 out of 49 teachers (63%) answered this question, whereas the others preferred to not answer it. We believe that participants who don't want to answer this question may not know how to use gamification or if there are gamified approaches available that can be used in their courses. For 9 out of 31 teachers answering the question, mainly timing constraints make that they are not interested in the introduction of gamification activities on their courses. Another 5 teachers, instead, are generally in favour for the introduction of these practices but did not provide any indications about the testing topics in which such gamification approaches could be more useful. Finally, the last 17 teachers provided more detailed indications, suggesting their liking for the introduction of gamification approaches in test case design and implementation and for practicing with testing strategies such as use case testing, scenario testing, equivalence partitioning, boundary analysis, white box testing, coverage-based testing and mutation-based testing.

Despite recent works in literature have evidenced the use of gamification to ignite intrinsic motivation of students to practice complex topics [53], and the use of gamification to improve the learning effectiveness [10], we understand teachers' thoughts regarding the use of gamification in testing courses. The introduction of gamification requires a detailed plan of the topics that will be practised with gamification and the topics that will be practised with other techniques, considering that a balance between gamification and other approaches is paramount to reap the benefits of gamification. Furthermore, as the use of gamification is still a promising technique to improve learning of complex topics, introducing gamification requires effort in terms of design and implementation of the gamified experience and a clear way to measure learning outcomes. We advocate that providing teachers with a repository with gamified approaches and guidelines on how to start using gamification will help to improve teachers' perceptions of the usefulness of using gamification in testing courses.

[14]Code Defenders, https://code-defenders.org/

Table 9. Summary of Answers to the question "Which testing topics taught in the course (if any) would need improved teaching materials?"

Testing Topic	# of answers
Acceptance testing	2
Coverage based testing	2
Functional testing	2
Fuzzing	1
GUI Testing	1
Integration testing	3
Testing methodology	6
Model-based testing	1
Mutation Testing	1
Specification-based testing	1
TDD	1
Teacher training	1
Test Automation	2
Test cases examples / exercises	2
Test execution environments	1
Generic	6
No specific requested improvements	14
No answers	2

4.6 RQ6 : Are the teachers satisfied with the available teaching materials? What are their main desiderata about them?

The final questions of the survey were aimed at collecting opinions about the quality of teaching as perceived by the teachers themselves and the opportunities to improve it. 18 out of 49 (37%) teachers said that they were satisfied by the teaching materials they used, while the rest (63%) said that it could be improved. Two of them suggested the need for more exercises and real use cases to be used instead of toy examples. We asked teachers for which testing topics they felt that there is lack of available teaching material. We received very fragmented answers, which we have summarized in Table 9. Although the majority of respondents asked for generic materials, a part of them explicitly mentioned specific testing topics such as Acceptance testing, Coverage-based testing, Fuzzing, GUI Testing, Integration testing, Model-based testing, Specification-based testing and TDD.

More specifically, we asked them what types of teaching Materials they would like to see improved. The most common answer was "solved examples and exercises" (requested by 19 out of 49 teachers). It is interesting to note that 6 out of 49 teachers requested new books, while other answers included real-world examples, systems to test, tutorials, videos and gamification examples.

Some teachers made more specific suggestions for specific materials or gamification tools. For example, some of them requested “a tool that shows the traceability from requirements to testing in real time”, “tools that provide a sequence of testing exercises with automatic feedback”, “real systems with known bugs that students can use to practice the techniques learned in the course”, “online resources students could explore with testing challenges”. Finally, one teacher suggested that “all the software testing teachers could be linked in some social network and share the material they use for the course (books, slides, etc.), as well as the solved examples and testing exercises, and how they do exams. Sharing knowledge, insights, and materials among teachers could be very useful. Such materials (the one to do exams) should not be available to everyone (students), but only to teachers”.

5 Discussion

In this Section we analyse the reported survey data in relation with current trends emerging from the scientific literature on software testing education, and outline the main insights and key takeaways.

RQ1: Organizational characteristics. The organizational characteristics of software testing courses found in the survey reflect existing trends in both the EU [74] [12] and other regions [56]. Commonalities include:

- The concentration of dedicated courses at the master's level.
- Limited integration of advanced testing topics in undergraduate programs.
- The use of practical, hands-on assessments in specialized courses.

These trends highlight shared challenges across regions, including a lack of alignment with industry needs and limited emphasis on testing at the undergraduate level. Addressing these issues will require global efforts to modernize curricula and expand the role of testing in software engineering education.

A possible reflection about this point is that software engineering education needs a global rethinking of curricula, ensuring testing concepts should be introduced earlier, emphasized more strongly, and aligned with professional practice. This is not just a local issue but a shared international challenge, calling for collaboration between academia and industry to modernize how testing is taught.

RQ2: Taught testing topics. Regarding the taught testing topics, the survey showed that:

- Teachers report using fragmented and inconsistent resources for software testing course design, with limited reliance on established frameworks like SWEBOK and ACM Curricula.
- Functional testing is almost universally taught in both dedicated software testing (ST) and broader software engineering courses (NST).
- Non-functional testing (e.g., performance, usability, security) is rarely included.
- Automated testing practices and model-based testing are the most commonly included teaching practices in ST courses, but NST courses primarily focus on manual and scripted testing.

Garousi et al. [42] and Barrett et al. [12] identify that teachers often lack sufficient guidance or standardized resources to implement effective testing education. The survey reflects this issue, emphasizing the need for better teacher support through standardized resources and professional development opportunities.

Previous studies, such as those by Garousi et al. [42], highlight that non-functional testing topics are often overlooked in academic curricula, despite their significance in industry. Studies by Valle et al. [76] in Brazil and Hanna [45] globally also identified that universities insufficiently address advanced testing topics, leaving a mismatch with industry needs. The survey confirms this trend, suggesting a persistent gap in teaching non-functional testing due to limited time, resources, or focus on industry needs.

The main takeaway is that education is lagging behind industry needs, delivering graduates not fully prepared for real-world testing challenges. Without intentional curriculum reform, stronger institutional guidance by standardized curricula and frameworks, stronger collaboration between academia and industry, this gap will persist globally, limiting the readiness of future software engineers.

RQ3: Adopted testing teaching practices. With respect to the six testing teaching practices investigated, our study showed:

- Test-Driven Development is the most frequently adopted software testing teaching practice. This is followed by an early introduction to testing, and the adaptive learning model.
- Few courses introduce active learning approaches like gamification or real-world project-based exercises, despite their potential to improve engagement and understanding.
- The least frequently adopted software testing teaching practices are case-based learning and automated assessment methods.

After a comparison with the current research trends we found that the prevalence of TDD is similar to the prevalence it has in current curricula. However, the research on adaptive learning models, case-based learning and an early introduction to software testing turned out to be less prevalent in research than in current curricula. Lastly, automated assessment methods turned out to be more widely researched than they have already been integrated with curricula.

The reflection that emerges here is that a closer alignment between research and teaching practice is needed. Research should pay more attention to the methods already used in classrooms, while educators should be supported in adopting evidence-based practices like automated assessments and active learning. This gap highlights the importance of bridging research and practice to modernize software testing education effectively.

RQ4: Adopted educational tools. The survey results show that educational tools are still underutilized in academic software testing courses. This is consistent with findings in the literature, such as those by Garousi et al. [40], which highlight the limited integration of specialized educational tools despite their potential benefits in engaging students and enhancing testing proficiency.

Most of the tools used by surveyed educators (e.g., Selenium, JUnit, Mockito) are industry-standard tools rather than ones specifically designed for educational purposes. Only a minority of instructors reported using tools like Bug Hunt, WebIDE, or Code Defenders, which are explicitly created to support interactive or gamified learning in testing education contexts. This low adoption reflects what Garousi et al. [43] and Pedreira et al. [59] identified as a gap between proposed solutions and real-world classroom uptake. Although many educational tools have been developed and described in the literature, few have achieved widespread use—possibly due to a lack of visibility, integration guidance, or teacher support.

The main takeaway is that further research is required to investigate strategies for increasing the adoption of educational tools in software testing courses. This may include developing frameworks for seamless integration of such tools into existing curricula, enhancing their visibility among instructors, and providing guidance or training to support effective classroom implementation.

RQ5: Adopted gamification approaches. The survey data showed that gamification was rarely used, with 86% of courses not using any gamification approach by teachers. In terms of intention to use gamification approaches, only 34% of teachers were in favour of doing so, and clearly indicated in which testing topics they would use gamification to improve students' practice.

Despite recent studies in the literature show that gamification can increase student engagement in learning software testing, with positive effects on performance [15] [69] [22] [33] [34], our survey highlights the need to promote the benefits of using gamification, to provide clear guidelines on how to start using gamified approaches, and to facilitate the use of freely available open-source gamified approaches for testing education.

The key insight is that gamification adoption is not primarily a matter of willingness but of support and resource availability, underscoring the importance of bridging research findings with actionable teaching strategies.

RQ6: Teacher satisfaction. The findings related to RQ6—teachers' satisfaction with current teaching materials—also align with recent literature. While many teachers reported using slides and exercises, only 41% of ST course instructors and 15% of NST instructors reported using textbooks, and a striking 27 different books were listed, with no single title dominating. This fragmentation points to the lack of consensus on core references, as also noted by Hanna [45] and Tramontana et al. [74].

This dispersion confirms what has been described in literature as a "lack of unified teaching material" or a "pedagogical patchwork" [43], which results in instructors building or curating their own resources—an effort that is often time-consuming and inconsistent.

Moreover, several respondents expressed desiderata for improved support materials, especially for practical exercises and case studies, echoing calls in the literature for high-quality, shareable resources that can help standardize and elevate software testing education.

Also worth noting is the limited adoption of real-world industrial examples, which the literature consistently advocates for as a means to bridge the gap between academic learning and professional readiness. Studies such as Elgrably et al. [32] and Doorn et al. [27] have emphasized the motivational and educational value of authentic, non-toy problems, yet their uptake remains rare in our sample.

The broader reflection is that advancing software testing education will require collective initiatives to standardize resources, promote the adoption of authentic case studies, and provide teachers with accessible, high-quality, and shareable materials. Projects like ENACTEST[15] [26, 54] explicitly target this issue by developing modular "teaching capsules" and aligning resources with industry needs. As a pioneering initiative, it provides a repository of current industry testing needs and a repository of educational materials (capsules) that teachers can easily use to transform the way testing is included in curricula, showing that coordinated efforts can help overcome fragmentation and better support both instructors and students.

6 Threats to validity

Our study design and data analysis involve several considerations regarding validity. Below, we discuss potential threats according to the classification of [81] and the steps taken to mitigate them.

6.1 Internal validity

To ensure that only relevant participants were included, we restricted the survey to teachers of courses that explicitly cover software testing topics. The survey was not released publicly, thereby reducing the likelihood of responses from individuals not meeting our inclusion criteria. Additionally, a dedicated question verified whether respondents were indeed teaching such a course. During data cleaning, two responses were excluded because the teachers declared that their courses did not include software testing.

6.2 Construct validity

The survey predominantly consisted of closed-form questions to reduce ambiguity and facilitate consistent interpretation. To further enhance clarity, we adopted the ISO/IEC/IEEE 29119-2022 standard to classify the subjects taught and provided precise definitions for potentially unfamiliar terms (e.g., Case-Based Learning). Open-ended answers were systematically coded by two authors, and inconsistencies (e.g., items labeled as both tools and frameworks) were

[15] https://enactest-project.eu

reconciled collaboratively by multiple authors. This process reduced the risk of misinterpretation and improved consistency.

6.3 Reliability

Data filtering, coding, and interpretation were performed collaboratively by multiple authors, which reduced the influence of individual bias and ensured consistency. All authors subsequently reflected on the interpretation of the results and the presentation of findings, reinforcing consensus. The anonymized dataset is publicly available, supporting transparency and replicability.

6.4 External validity

The surveyed teachers were exclusively affiliated with public universities in Western Europe. While this provides a degree of homogeneity in terms of educational structures (e.g., degree levels, ECTS system), it limits the generalizability of the results to other regions and institutional settings. To partially mitigate this threat, we made the anonymized dataset available online, enabling other researchers to replicate or extend the study. Future work should extend the survey to a more geographically and institutionally diverse set of participants to validate and generalize the findings at a global scale.

7 Conclusions

This study provides a contemporary snapshot of university-level software testing education in Western Europe, offering insights directly from instructors rather than relying on secondary data or course descriptions. By capturing organizational aspects, taught topics, teaching practices, and perceptions of available resources, our study complements and updates earlier surveys, whose data often predate recent curricular and technological changes.

Our study confirmed prior findings on the testing topics covered in courses but also showed that ACM and SWEBOK recommendations are rarely adopted. Teaching practices are dominated by Test Driven Development, with adaptive and case-based learning used more sporadically, while early introduction of testing and the use of industry examples remain uncommon. The adoption of educational tools is limited, mainly due to the absence of widely supported platforms and materials, and gamification approaches are rarely employed owing to insufficient guidance for their effective integration.

These findings underline persistent challenges in bridging academic practices with industry demands, but also reveal opportunities: the increasing adoption of TDD and automated assessment suggests readiness to integrate more innovative practices if suitable materials and guidance are available. Likewise, the lack of widely supported platforms, materials, and gamification approaches highlights the need for future work on developing improved teaching resources that directly address the needs identified by the educators in our study.

Our publicly available dataset can serve as a foundation for comparative analyses, longitudinal studies, and the design of interventions aimed at improving alignment, tool adoption, and the sharing of resources across institutions. Building on this evidence, future work should both foster a more consistent and industry-relevant approach to software testing education and include a redesign of the survey to capture the evolution of teaching practices, particularly in response to the proliferation of disruptive technologies such as AI.

References

[1] Rahul Agarwal, Stephen H. Edwards, and Manuel A. Pérez-Quiñones. 2006. Designing an adaptive learning module to teach software testing. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA) *(SIGCSE '06)*. Association for Computing Machinery, New York, NY, USA, 259–263. doi:10.1145/1121341.1121420

[2] Anthony Allowatt and Stephen H. Edwards. 2005. IDE Support for test-driven development and automated grading in both Java and C++. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange* (San Diego, California) *(eclipse '05)*. Association for Computing Machinery, New York, NY, USA, 100–104. doi:10.1145/1117696.1117717

[3] Domenico Amalfitano, Ana C. R. Paiva, Alexis Inquel, Luís Pinto, Anna Rita Fasolino, and René Just. 2022. How do Java mutation tools differ? *Commun. ACM* 65, 12 (Nov. 2022), 74–89. doi:10.1145/3526099

[4] Paul Ammann and Jeff Offutt. 2017. *Introduction to Software Testing* (2nd ed.). Cambridge University Press, Cambridge, UK. doi:10.1017/9781316771273

[5] Maurício Aniche. 2022. *Effective Software Testing: A Developer's Guide*. Manning Publications, Shelter Island, New York.

[6] Maurício Aniche, Felienne Hermans, and Arie van Deursen. 2019. Pragmatic Software Testing Education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 414–420. doi:10.1145/3287324.3287461

[7] Maurício Aniche, Frank Mulder, and Felienne Hermans. 2021. Grading 600+ Students: A Case Study on Peer and Self Grading. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 211–220. doi:10.1109/ICSE-SEET52601.2021.00031

[8] Baris Ardic and Andy Zaidman. 2023. Hey Teachers, Teach Those Kids Some Software Testing. In *2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG)*. 9–16. doi:10.1109/SEENG59157.2023.00007

[9] Tara Astigarraga, Eli M. Dow, Christina Lara, Richard Prewitt, and Maria R. Ward. 2010. The Emerging Role of Software Testing in Curricula. In *2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments*. 1–26. doi:10.1109/TEE.2010.5508833

[10] Shurui Bai, Khe Foon Hew, and Biyun Huang. 2020. Does gamification improve student learning outcome? Evidence from a meta-analysis and synthesis of qualitative data in educational contexts. *Educational Research Review* 30 (2020). doi:10.1016/j.edurev.2020.100322

[11] Pedro Luis Saraiva Barbosa, Rafael Augusto Ferreira do Carmo, João P. P. Gomes, and Windson Viana. 2023. Adaptive learning in computer science education: A scoping review. *Education and Information Technologies* 29, 8 (Sept. 2023), 9139–9188. doi:10.1007/s10639-023-12066-z

[12] Ayodele A. Barrett, Eduard Paul Enoiu, and Wasif Afzal. 2023. On the Current State of Academic Software Testing Education in Sweden. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 397–404. doi:10.1109/ICSTW58534.2023.00073

[13] Kent Beck. 2002. *Test Driven Development: By Example*. Addison-Wesley, Boston, MA.

[14] Jonathan Bell, Swapneel Sheth, and Gail Kaiser. 2011. Secret ninja testing with HALO software engineering. In *Proceedings of the 4th International Workshop on Social Software Engineering* (Szeged, Hungary) *(SSE '11)*. Association for Computing Machinery, New York, NY, USA, 43–47. doi:10.1145/2024645.2024657

[15] Raquel Blanco, Manuel Trinidad, María José Suárez-Cabal, Alejandro Calderón, Mercedes Ruiz, and Javier Tuya. 2023. Can gamification help in software testing education? Findings from an empirical study. *Journal of Systems and Software* 200 (2023), 111647. doi:10.1016/j.jss.2023.111647

[16] Gianluigi Caldiera, Victor Basili, and Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* (1994), 528–532.

[17] Felix Cammaerts, Porfirio Tramontana, Ana C. R. Paiva, Nuno Flores, Fernando Pastor Ricós, and Monique Snoeck. 2024. Exploring students' opinion on software testing courses. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 570–579. doi:10.1145/3661167.3661276

[18] Stephen Cass. 2024. The Top Programming Languages 2024. *IEEE Spectrum* (August 2024). https://spectrum.ieee.org/top-programming-languages-2024 Accessed: Dec 2024.

[19] Kelsey Hood Cattaneo. 2017. Telling Active Learning Pedagogies Apart: From Theory to Practice. *Journal of New Approaches in Educational Research* 6, 2 (2017), 144–152. doi:10.7821/naer.2017.7.237

[20] T.Y. Chen and Pak-Lok Poon. 2004. Experience with teaching black-box testing in a computer science/software engineering curriculum. *IEEE Transactions on Education* 47, 1 (2004), 42–50. doi:10.1109/TE.2003.817617

[21] Li-Ren Chien, Daniel J. Buehrer, Chin-Yi Yang, and Chyong-Mei Chen. 2008. An evaluation of TDD training methods in a programming curriculum. In *2008 IEEE International Symposium on IT in Medicine and Education*. 660–665. doi:10.1109/ITME.2008.4743948

[22] Gabriela Martins de Jesus, Leo Natan Paschoal, Fabiano Cutigi Ferrari, and Simone R. S. Souza. 2019. Is It Worth Using Gamification on Software Testing Education? An Experience Report. In *Proceedings of the XVIII Brazilian Symposium on Software Quality* (Fortaleza, Brazil) *(SBQS '19)*. Association for Computing Machinery, New York, NY, USA, 178–187. doi:10.1145/3364641.3364661

[23] Daniel de Paula Porto, Gabriela Martins de Jesus, Fabiano Cutigi Ferrari, and Sandra Camargo Pinto Ferraz Fabbri. 2021. Initiatives and challenges of using gamification in software engineering: A Systematic Mapping. *Journal of Systems and Software* 173 (2021), 110870. doi:10.1016/j.jss.2020.110870

[24] Chetan Desai, David Janzen, and Kyle Savage. 2008. A survey of evidence for test-driven development in academia. *SIGCSE Bull.* 40, 2 (June 2008), 97–101. doi:10.1145/1383602.1383644

[25] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. 2011. Gamification. using game-design elements in non-gaming contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI EA '11)*. Association for Computing

Machinery, New York, NY, USA, 2425–2428. doi:10.1145/1979742.1979575

[26] Niels Doorn, Fernando Pastor Ricós, Beatriz Marín, and Tanja Vos. 2025. Alianza Europea de Innovación para la Educación del Testing-Proyecto ENACTEST. In *Congresso Ibero-Americano em Engenharia de Software (CIbSE)*. SBC, 367–370.

[27] Niels Doorn, Tanja E.J. Vos, and Beatriz Marín. 2023. Towards understanding students' sensemaking of test case design. *Data and Knowledge Engineering* 146 (2023), 102199. doi:10.1016/j.datak.2023.102199

[28] Thomas Dvornik, David S. Janzen, John Clements, and Olga Dekhtyar. 2011. Supporting introductory test-driven labs with WebIDE. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*. 51–60. doi:10.1109/CSEET.2011.5876137

[29] Stephen H. Edwards. 2003. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.* 3, 3 (Sept. 2003), 1–es. doi:10.1145/1029994.1029995

[30] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable. In *29th International Conference on Software Engineering (ICSE'07)*. 688–697. doi:10.1109/ICSE.2007.23

[31] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. 2007. Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable. In *29th International Conference on Software Engineering (ICSE'07)*. 688–697. doi:10.1109/ICSE.2007.23

[32] Isaac Souza Elgrably and Sandro Ronaldo Bezerra de Oliveira. 2021. A diagnosis on software testing education in the Brazilian Universities. In *2021 IEEE Frontiers in Education Conference (FIE)* (Lincoln, NE, USA). IEEE Press, 1–8. doi:10.1109/FIE49875.2021.9637305

[33] Anna Rita Fasolino, Caterina Maria Accetto, and Porfirio Tramontana. 2024. Testing Robot Challenge: A Serious Game for Testing Learning. In *Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation* (Vienna, Austria) *(Gamify 2024)*. Association for Computing Machinery, New York, NY, USA, 26–29. doi:10.1145/3678869.3685686

[34] Anna Rita Fasolino and Porfirio Tramontana. 2024. Test Smells Learning by a Gamification Approach. In *Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation, Gamify 2024, Vienna, Austria, 17 September 2024*. ACM, 30–33. doi:10.1145/3678869.3685687

[35] Gordon Fraser. 2019. Code Defenders: A Mutation Testing Game. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 1289. doi:10.1145/3287324.3293753

[36] Gordon Fraser, Alessio Gambi, Marvin Kreis, and José Miguel Rojas. 2019. Gamifying a Software Testing Course with Code Defenders. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 571–577. doi:10.1145/3287324.3287471

[37] Gordon Fraser, Alessio Gambi, and José Miguel Rojas. 2018. A Preliminary Report on Gamifying a Software Testing Course with the Code Defenders Testing Game. In *Proceedings of the 3rd European Conference of Software Engineering Education* (Seeon/ Bavaria, Germany) *(ECSEE '18)*. Association for Computing Machinery, New York, NY, USA, 50–54. doi:10.1145/3209087.3209103

[38] Yujian Fu and Peter J. Clarke. 2016. Gamification-based cyber-enabled learning environment of software testing. *ASEE Annual Conference and Exposition, Conference Proceedings* 2016-June (2016).

[39] Yujian Fu, Peter J. Clarke, and Nelson Barnes. 2016. Integrating software testing to CS curriculum using WRESTT-CyLE. *ASEE Annual Conference and Exposition, Conference Proceedings* 2016-June (2016).

[40] Kirti Garg, Ashish Sureka, and Vasudeva Varma. 2015. A Case Study on Teaching Software Engineering Concepts using a Case-Based Learning Environment.. In *QuASoQ/WAWSE/CMCE@ APSEC*. 71–78. https://ceur-ws.org/Vol-1519/paper15.pdf

[41] Kirti Garg and Vasudeva Varma. 2007. A Study of the Effectiveness of Case Study Approach in Software Engineering Education. In *Proceedings of the 20th Conference on Software Engineering Education & Training (CSEET '07)*. IEEE Computer Society, USA, 309–316. doi:10.1109/CSEET.2007.8

[42] Vahid Garousi, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer. 2019. Aligning software engineering education with industrial needs: A meta-analysis. *J. Syst. Softw.* 156, C (Oct. 2019), 65–83. doi:10.1016/j.jss.2019.06.044

[43] Vahid Garousi, Austen Rainer, Per Lauvås, and Andrea Arcuri. 2020. Software-testing education: A systematic literature mapping. *Journal of Systems and Software* 165 (2020), 110570. doi:10.1016/j.jss.2020.110570

[44] Alessio Gaspar and Sarah Langevin. 2007. Restoring "coding with intention" in introductory programming courses. In *Proceedings of the 8th ACM SIGITE Conference on Information Technology Education* (Destin, Florida, USA) *(SIGITE '07)*. Association for Computing Machinery, New York, NY, USA, 91–98. doi:10.1145/1324302.1324323

[45] Samer Hanna. 2024. Analysis of the gap between software testing courses at universities and the needed skills by industry. *Int. J. Learn. Technol.* 19, 2 (Jan. 2024), 203–242. doi:10.1504/ijlt.2024.139038

[46] Hadi Hemmati. 2015. How Effective Are Code Coverage Criteria?. In *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Vancouver, BC, Canada, 151–156. doi:10.1109/QRS.2015.30

[47] Pedro Henrique Dias Valle, Armando Maciel Toda, Ellen Francine Barbosa, and José Carlos Maldonado. 2017. Educational games: A contribution to software testing education. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–8. doi:10.1109/FIE.2017.8190470

[48] Nurul Jamal, Dayang Jawawi, Rohayanti Hassan, and Irsyad Riadz. 2020. Adaptive learning in computing education: A systematic mapping study. In *IOP Conference Series: Materials science and engineering*, Vol. 864. IOP Publishing. doi:10.1088/1757-899X/864/1/012069

[49] David Janzen and Hossein Saiedian. 2008. Test-driven learning in early programming courses. *SIGCSE Bull.* 40, 1 (March 2008), 532–536. doi:10.1145/1352322.1352315

[50] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. doi:10.1109/TSE.2010.62

[51] Stephen H. Edwards Kevin Buffardi. 2012. Impacts of Teaching Test-Driven Development to Novice Programmers. *International Journal of Information and Computer Science* 6 (September 2012), 135–143. https://ia902303.us.archive.org/17/items/IJICS041/IJICS041.pdf

[52] Daniel E. Krutz, Samuel A. Malachowsky, and Thomas Reichlmayr. 2014. Using a real world project in a software testing course. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 49–54. doi:10.1145/2538862.2538955

[53] Beatriz Marín. 2023. Gamification to Ignite Learning in Modern Times (Keynote). In *Proceedings of the 2nd International Workshop on Gamification in Software Development, Verification, and Validation* (San Francisco, CA, USA) *(Gamify 2023)*. Association for Computing Machinery, New York, NY, USA, 1. doi:10.1145/3617553.3634752

[54] Beatriz Marín, Tanja E. J Vos, Ana CR Paiva, Anna Rita Fasolino, and Monique Snoeck. 2022. ENACTEST-European Innovation Alliance for Testing Education.. In *RCIS Workshops*. https://ceur-ws.org/Vol-3144/RP-paper5.pdf

[55] Will Marrero and Amber Settle. 2005. Testing first: emphasizing testing in early programming courses. *SIGCSE Bull.* 37, 3 (June 2005), 4–8. doi:10.1145/1151954.1067451

[56] Silvana M. Melo, Veronica X. S. Moreira, Leo Natan Paschoal, and Simone R. S. Souza. 2020. Testing Education: A Survey on a Global Scale. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 554–563. doi:10.1145/3422392.3422483

[57] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (June 2022), 40 pages. doi:10.1145/3513140

[58] Leo Natan Paschoal and Simone do Rocio Senger de Souza. 2018. A Survey on Software Testing Education in Brazil. In *Proceedings of the XVII Brazilian Symposium on Software Quality* (Curitiba, Brazil) *(SBQS '18)*. Association for Computing Machinery, New York, NY, USA, 334–343. doi:10.1145/3275245.3275289

[59] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. 2015. Gamification in software engineering – A systematic mapping. *Information and Software Technology* 57 (2015), 157–168. doi:10.1016/j.infsof.2014.08.007

[60] Ita Richardson and Yvonne Delaney. 2009. Problem Based Learning in the Software Engineering Classroom. In *2009 22nd Conference on Software Engineering Education and Training*. 174–181. doi:10.1109/CSEET.2009.34

[61] Lilian Passos Scatalon, Jeffrey C. Carver, Rogério Eduardo Garcia, and Ellen Francine Barbosa. 2019. Software Testing in Introductory Programming Courses: A Systematic Mapping Study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 421–427. doi:10.1145/3287324.3287384

[62] Lilian Passos Scatalon, Rogerio Eduardo Garcia, and Ellen Francine Barbosa. 2020. Teaching Practices of Software Testing in Programming Education. *Proceedings - Frontiers in Education Conference, FIE* 2020-October (2020). doi:10.1109/FIE44824.2020.9274256

[63] Zalia Shams. 2013. Automated assessment of students' testing skills for improving correctness of their code. In *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity* (Indianapolis, Indiana, USA) *(SPLASH '13)*. Association for Computing Machinery, New York, NY, USA, 37–40. doi:10.1145/2508075.2508078

[64] Robert M. Siegfried, Katherine G. Herbert-Berger, Kees Leune, and Jason P. Siegfried. 2021. Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning. In *2021 16th International Conference on Computer Science and Education (ICCSE)*. 407–412. doi:10.1109/ICCSE51940.2021.9569444

[65] Mateus Silva, Ana CR Paiva, and Alexandra Mendes. 2025. GAMFLEW: serious game to teach white-box testing. *Software Quality Journal* 33, 1 (2025), 5. doi:10.1007/s11219-024-09706-z

[66] Rebecca Smith, Terry Tang, Joe Warren, and Scott Rixner. 2017. An Automated System for Interactively Learning Software Testing *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 98–103. doi:10.1145/3059009.3059022

[67] Ian Sommerville. 2015. *Software Engineering* (10th ed.). Pearson, Boston, MA.

[68] Jaime Spacco and William Pugh. 2006. Helping students appreciate test-driven development (TDD). In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA) *(OOPSLA '06)*. Association for Computing Machinery, New York, NY, USA, 907–913. doi:10.1145/1176617.1176743

[69] Philipp Straubinger and Gordon Fraser. 2024. Gamifying a Software Testing Course with Continuous Integration. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (Lisbon, Portugal) *(ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 34–45. doi:10.1145/3639474.3640054

[70] Joseph Timoney, Stephen Brown, and Deshi Ye. 2008. Experiences in Software Testing Education: Some Observations from an International Cooperation. In *2008 The 9th International Conference for Young Computer Scientists*. 2686–2691. doi:10.1109/ICYCS.2008.209

[71] Elena Tiukhova, Charlotte Verbruggen, Bart Baesens, and Monique Snoeck. 2024. Learning analytics tells: Know your basics and go to class. In *CEUR Workshop Proceedings*, Vol. 3618. CEUR Workshop Proceedings. https://ceur-ws.org/Vol-3618/scme_paper_2.pdf

[72] Saurabh Tiwari, Veena Saini, Paramvir Singh, and Ashish Sureka. 2018. A Case Study on the Application of Case-Based Learning in Software Testing. In *Proceedings of the 11th Innovations in Software Engineering Conference* (Hyderabad, India) *(ISEC '18)*. Association for Computing Machinery, New York, NY, USA, Article 11, 5 pages. doi:10.1145/3172871.3172881

[73] Hallvard Trætteberg and Trond Aalberg. 2006. JExercise: a specification-based and test-driven exercise support plugin for Eclipse. In *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology EXchange* (Portland, Oregon, USA) *(eclipse '06)*. Association for Computing Machinery, New York, NY, USA, 70–74. doi:10.1145/1188835.1188850

[74] Porfirio Tramontana, Beatriz Marín, Ana C. R. Paiva, Alexandra Mendes, Tanja E. J. Vos, Domenico Amalfitano, Felix Cammaerts, Monique Snoeck, and Anna Rita Fasolino. 2024. State of the Practice in Software Testing Teaching in Four European Countries. (2024), 59–69. doi:10.1109/ICST60714.2024.00015

[75] Christos Troussas, Akrivi Krouska, and Cleo Sgouropoulou. 2021. A Novel Teaching Strategy Through Adaptive Learning Activities for Computer Programming. *IEEE Transactions on Education* 64, 2 (2021), 103–109. doi:10.1109/TE.2020.3012744

[76] Pedro Henrique Dias Valle, Ellen Francine Barbosa, and José Carlos Maldonado. 2015. CS curricula of the most relevant universities in Brazil and abroad: Perspective of software testing education. In *2015 International Symposium on Computers in Education (SIIE)*. 62–68. doi:10.1109/SIIE.2015.7451649

[77] Steven Van Goidsenhoven, Daria Bogdanova, Galina Deeva, Seppe vanden Broucke, Jochen De Weerdt, and Monique Snoeck. 2020. Predicting student success in a blended learning environment. In *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge* (Frankfurt, Germany) *(LAK '20)*. Association for Computing Machinery, New York, NY, USA, 17–25. doi:10.1145/3375462.3375494

[78] H. Washizaki. 2024. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. *ISO/IEC/IEEE 29119-1:2022(E)* (2024). www.swebok.org

[79] Dee A. B. Weikle, Michael O. Lam, and Michael S. Kirkpatrick. 2019. Automating Systems Course Unit and Integration Testing: Experience Report. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 565–570. doi:10.1145/3287324.3287502

[80] Wu Wen, Jiahui Sun, Ya Li, Peng Gu, and Jianfeng Xu. 2019. Design and Implementation of Software Test Laboratory Based on Cloud Platform. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 138–144. doi:10.1109/QRS-C.2019.00039

[81] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[82] Michal Young and Mauro Pezze. 2005. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[83] Tamara Zivkovic, Drazen Draskovic, and Bosko Nikolic. 2023. Learning environments in software testing education: An overview. *Computer Applications in Engineering Education* 31, 6 (2023), 1497–1521. doi:10.1002/cae.22657