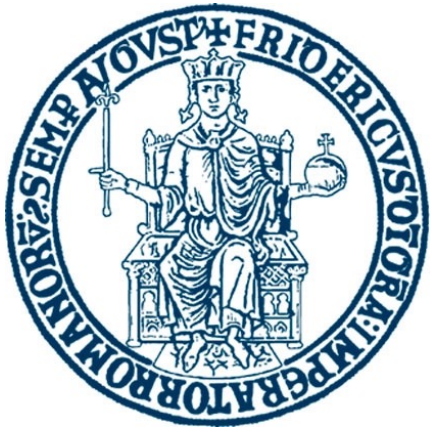


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Beyond the Class: A look into current trends in software testing education

Anna Rita Fasolino

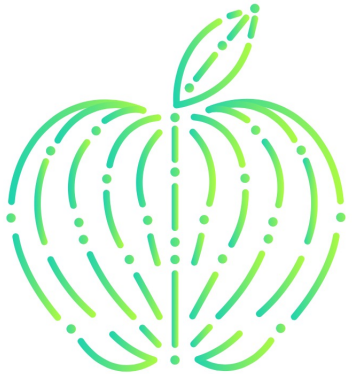
University of Napoli Federico II, Napoli, Italy

fasolino@unina.it

Funded by the European Union

ERASMUS plus Project 2022-2025

ENACTEST



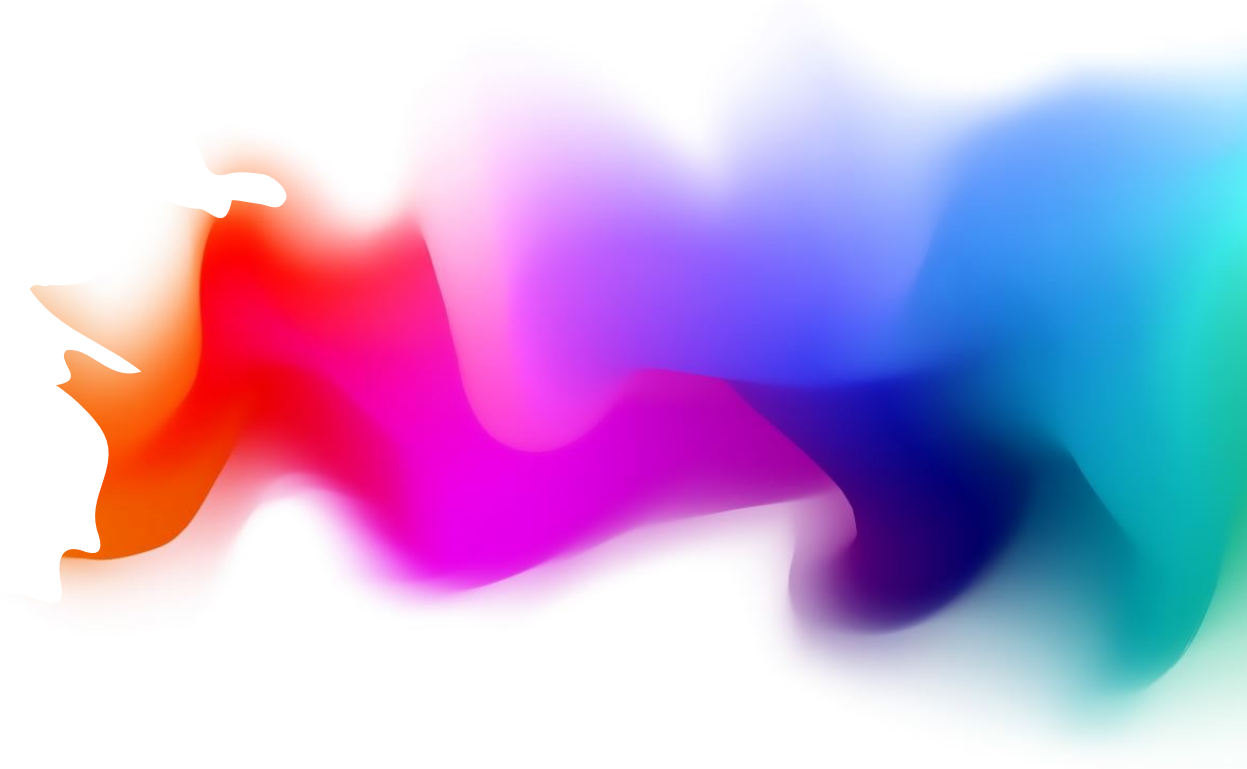
Motivation

- Software testing is indispensable in software development, yet often overlooked, contributing to a shortage of expertise in the software industry.
- Becoming an experienced software tester requires understanding many strategies for writing high-quality test cases and a significant amount of practice.
- Despite efforts to improve teaching approaches at the university level, **many challenges** persist for better preparing students for their future careers.

Questions we will try to answer in this talk

- What are current **challenges** in software testing education?
- What are the **solutions** proposed to address these challenges?
- What is the **state of the practice** in Teaching Testing at University level?
- What are the **future perspectives** in Software Testing education ?

Challenges in Software Testing Education



Challenges reported in the literature

- Challenges reported by a SLR by **Scatalon** et al. (2017) [1]
  - of integrating software testing into introductory programming courses (based on 158 papers)
- Challenges described by **Delgado-Perez** et al. (2021) [2]
  - of instructing students in the use of testing techniques and the importance of software testing within the software development
- Challenges analysed in the Systematic Mapping study by **Garousi** et al. (2020) [3]
  - Nine categories of challenges emerged from more than 200 papers

[1] Scatalon, L.P. , Barbosa, E.F. , Garcia, R.E. , 2017. **Challenges to integrate software testing into introductory programming courses**. In: IEEE Frontiers in Education Conference, pp. 1–9 .
[2] P. Delgado-Pérez, et al. "**Mutation Testing and Self/Peer Assessment: Analyzing their Effect on Students in a Software Testing Course**," *2021 IEEE/ACM 43rd Int. Conf. on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Madrid, ES, 2021, pp. 231-240.
[3] Vahid Garousi, et al.: **Software-testing education: A systematic literature mapping.** J. Syst. Softw. 165: 110570 (2020)

Challenges related to Students and Educators

1.Testing often not well accepted among students, low motivation, tedious, boring

- Students do not derive **a great deal of satisfaction** from **exposing flaws** in their own programs.
- Students perceive testing as **not important**, **boring and repetitive**, and do not acquire the practice and experience that software testing requires.
- The programs used in software testing courses are often **simple, only toy programs** that are difficult to stimulate students' interests and enthusiasms.
- The traditional **pedagogical approaches** are not sufficient to make students motivated to write unit tests as they code.

2.Tool-related challenges

- Technical **issues with testing tools,** expecially for beginners
- The **lack of good supporting tools for teaching** testing practices in an introductory level
- Good support for *unit* testing, but more complex testing often lacks good, easy-to-use tools
- **Testing** and **Automated** testing requires a good knowledge of programming
- **Difficulties in teaching xUnit-style unit testing** frameworks in CS1
- Difficulties with reproducing in class the development/testing **platforms and pipelines** which are vital for working effectively in practice

3. Increased cognitive load for learning testing

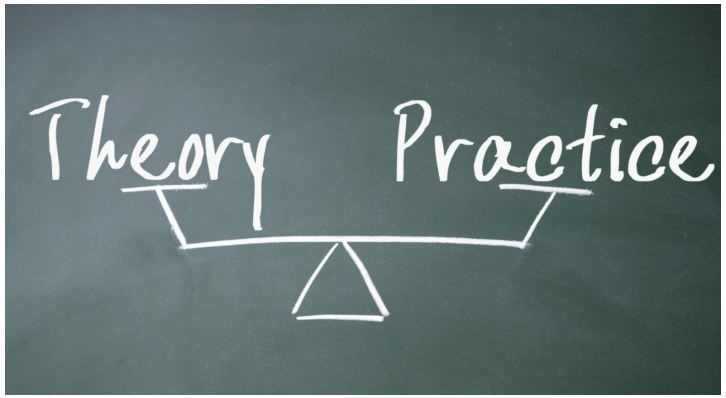
- **Teaching** software **testing skills for first year students** CS courses can be particularly challenging
  - students have to deal with peculiarities of the **specific techniques and tools** for software testing
  - They have to learn **additional syntax** in order to express their test cases
  - teaching relies heavily on **experiential learning**

- Teaching **Test-Driven Development (TDD)** increases technical and cognitive load for the students.
  - TDD **requires a change in thinking** and does not come naturally to all students
  - TDD cycle is difficult to practice

Challenges related to Educators

4. Suitable course design: Alignment with industry needs

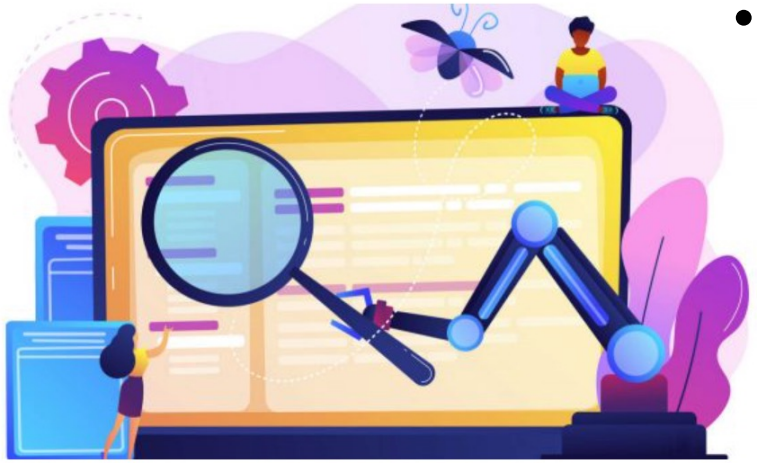
- Software testing education is considered **too much theoretical** with a lack of practical application scenarios (Scatalon –survey with Practitioners 2018)
- There's a **gap between book knowledge (theory) and practice**
- **Gap** between **formal education** and **industry practice**s
- Disconnection between theory and practice leads to **less interest by students**



- From the educator's perspective, it is hard to keep a testing course up-to-date with the **novelties** of the field, as well as to come up with **exercises that are realistic**
- **Incorporating real-world industrial testing projects** in software testing courses would be necessary but is challenging

5. Suitable course design: Issue of task "scale" / complexity

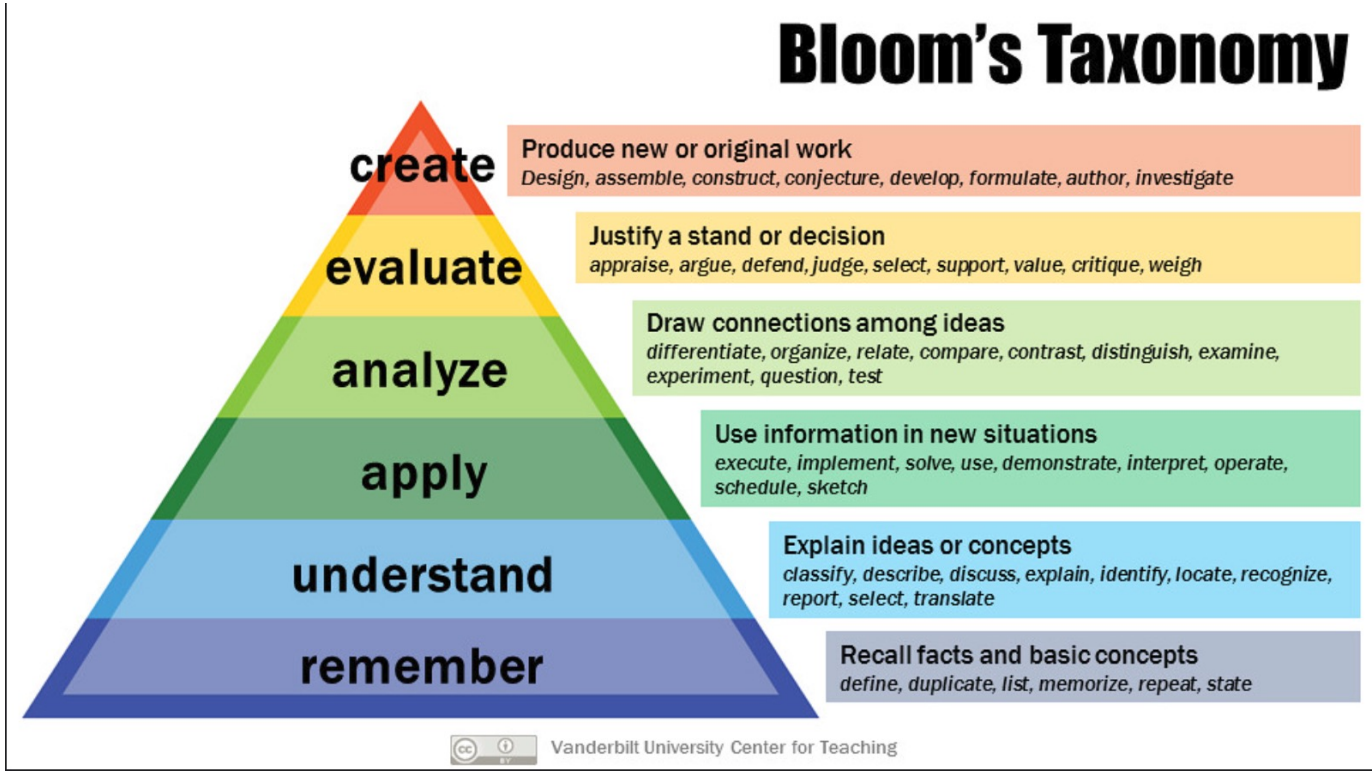
- Setting the **appropriate complexity** of testing tasks for students is challenging
- Students perceive test writing to be **irrelevant** and more costly than beneficial **due to the small size** of the programming task



- Developing software tests for more complex programs that have **significant graphical user interfaces** is beyond the abilities of typical students
- Students in the introductory CS1 programming course can have a difficult time developing and implementing **sufficient test cases**

6. Suitable course design: pedagogical issues …

- Challenges of developing **higher-order thinking** (Bloom's taxonomy)
- Traditional Testing lectures would be most appropriate for conveying factual and conceptual knowledge at the **remembering and understanding** levels of the Bloom's Taxonomy
- Students of testing also need to learn how to **analyze** situations and problems, **apply** techniques, and **evaluate** their own work and the work of their peers



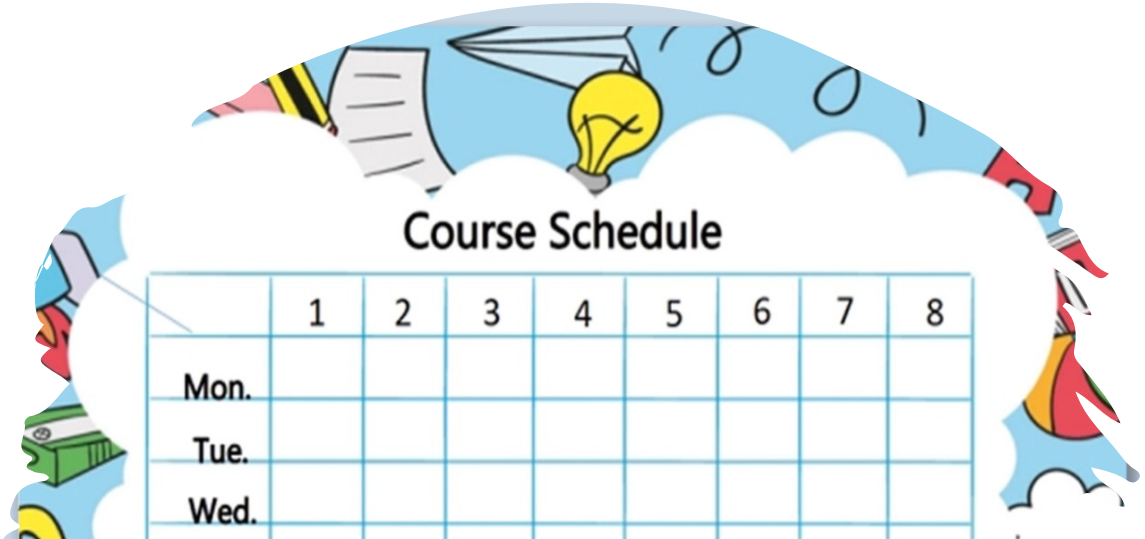
Appropriate course materials and teaching strategies in the classroom are needed!

C. Kaner and S. Padmanabhan, "Practice and Transfer of Learning in the Teaching of Software Testing," *20th Conference on Software Engineering Education & Training (CSEET'07)*, Dublin, Ireland, 2007, pp. 157-166

6. Suitable course design: pedagogical issues

- Educators face the fact that some **testing topics** are **not conceptually straightforward**, not easy to demonstrate and generalize, and are not all available in a single textbook
- The **software testing laboratory is limited**, and so classroom teaching and practice are not closely related

- Students often struggle with the **concept of testing to find bugs** rather than just testing to show that their software is operating perfect on a given set of inputs.
- Students **need frequent, concrete feedback** on how to improve their performance at many points throughout their development of a solution, rather than just once at the end of an assignment.



7. Time/ Resource Requirements

- **Not enough time** to teach testing in programming and software engineering courses
- **Too many topics** to be covered in software testing courses
- It is challenging to **evaluate thousands of assignments** within limited time
- Due to time constraints, it may not be feasible to **assess test cases** .

8.Challenges w.r.t. integrating software testing in other courses

- **Two options** in the curricula design:
  - Offering a **dedicated course** on software testing? *(more time and resources to cover this complex subject in more detail, but often considered tedious...)*
  - **Integrating software testing** in other core courses ?

- Due to several reasons (limited resources, low motivation, etc.) the teaching of software testing is often spread across several programming courses!

Challenges of Integrating Software Testing in other courses

- Decisions on sequencing and course content.
- Software testing requires students to have experience in programming.
- Students' development experience may be insufficient for them to understand TDD.
- How to provide an appropriate feedback and to evaluate the student's performance in integrated courses?

9. Not easy to assess students work

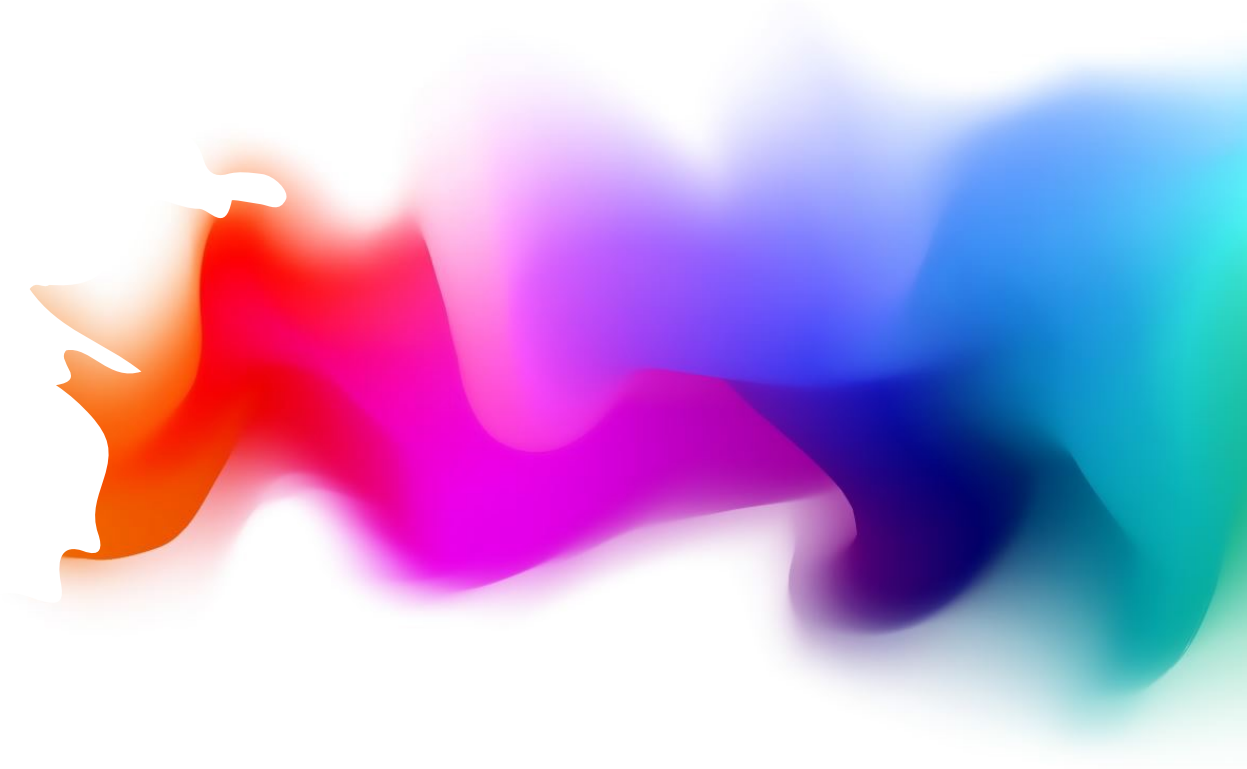
- How to **effectively and efficiently evaluate** if a student has accomplished the stated outcome?
- Current assessment techniques used in automated grading tools for evaluating student-written software tests are imperfect.
- In programming courses, instructors need to assess program correctness first. Due to time constraints, it **may not be feasible to assess** test cases too.

- **Code coverage** alone is not a satisfactory measure for test quality
- Far less attention has been given to assessing the **quality of the tests** produced by the students themselves
- Advanced testing techniques like **mutation testing** could help in such regard, but, unfortunately, it is not so well known outside of test specialists, and tool support is still rather unsatisfactory

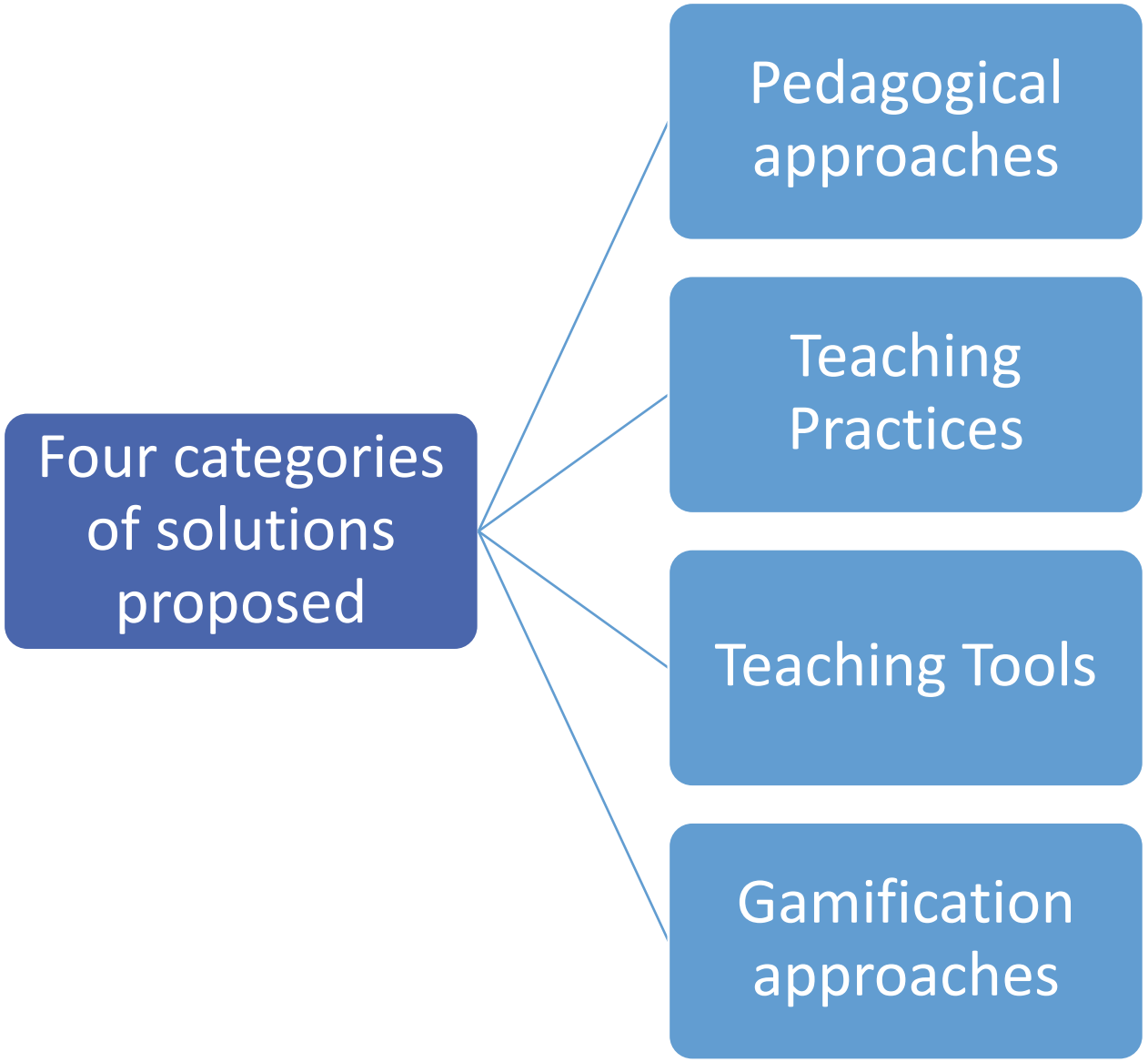
A synthesis of challenges

#	Challenge
1	Testing often not well accepted among students, low motivation, tedious, boring
2	Tool-related challenges
3	Increased cognitive load for learning testing
4	Suitable course design: Alignment with industry needs
5	Suitable design of the course: Issue of "scale" / complexity
6	Suitable design of the course: other issues
7	Time/ Resource Requirements
8	Challenges w.r.t. integrating software testing in other courses
9	Not easy to assess students work

Possible Solutions



How have been these challenges addressed in the literature?



Pedagogical Approaches

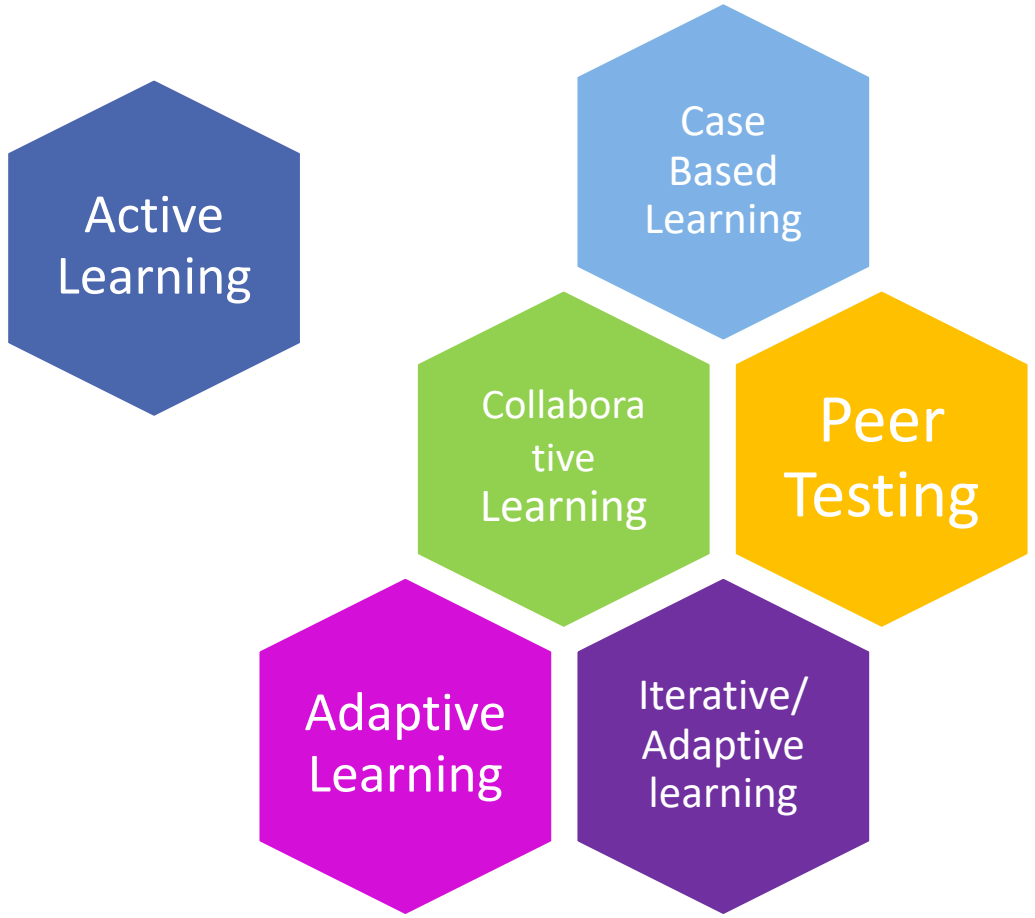
- Software Testing is an experimental activity, so it should be mostly practical, and **Active** learning approaches should be preferred to **Passive** ones
- Active learning can essentially be defined as "*students doing things and thinking about what they are doing*" (Bonwell and Eison, 1991).
- Active vs Passive Learning
  - Active learning can be facilitated through a range of teaching approaches that encourage learners to actively engage with course materials, one another, and/or with lecturers. It contrasts with passive learning of just reading or listening – the 'talking head' style of instruction.

A classification of Active learning activities

(*https://itali.uq.edu.au/teaching-guidance/teaching-practices/active-learning*)

Active Learning Activity	Description
Project-based learning	PBL is a student-centred methodology that engages students in developing critical thinking through undertaking authentic, meaningful projects.
Case/scenario/problem/ inquiry-based learning	require students to apply their disciplinary knowledge, critical thinking and problem-solving skills in a safe, real-world context.
Reflective learning	Reflective learning develops students' critical thinking skills by analysing experiences to improve future performance.
Collaborative learning	CL encompasses activities ranging from classroom discussions to problem-solving in groups, to working in teams
Experiential learning	The learner is an active participant in the educational process, and learning is achieved through a continuous cycle of inquiry, reflection, analysis and synthesis
In-class active learning activities	pedagogical strategies and methods you can utilise in your teaching to spark engagement or be part of extended and connected learning activities across a class or course (ex. One-minute paper, Peer-instruction, Think-Pair-Share…)

Some Active Learning and other pedagogical strategies described in the literature



Teaching Practices

- Teaching practices that have shown their effectiveness in helping teachers and students to improve testing educational processes

- **Four types of practice** extracted from the literature:
  - Types of Software to be Tested
  - Moments to start talking about Testing in the curricula
  - Ways to evaluate students' testing learning
  - Practical solutions for better teaching testing

Recommended Types of Software to be tested by students

Using free or open-source projects (F/OSS), rather than toy or student-made software
Students write tests for code produced by fellow students
Testing on real world applications, in collaboration with a real industrial company
Testing realistic software

The type of software to be tested is able to affect the motivation and interest of the students

Recommended moments to start talking of Testing

Objects first, tests second
Applying the Testing Before Coding (TBC) method in introductory programming courses
Early testing

Early introducing software testing and TDD since CS1 and CS2 courses can help students become familiar with testing concepts and motivated to write unit tests as they code

Recommended ways to evaluate students' testing learning

Providing students with automatic, concrete, incremental feedback on performance
Using Automated assessment of students' testing skills for improving correctness of their code
Better approaches to evaluate student-written tests, beyond Code Coverage
Subjective Self and Peer assessments
Rapid automated feedback on software tests for complex projects

Recommended practices for teaching testing

Using test-driven development in the classroom
Unit testing using the Given, When, Then pattern
Integrating automated unit testing practices in an introductory programming course
Teaching testing using analysis of Known Bugs, bug-hunting gamification
Using industry-strength tool

Teaching Tools

- Tools, simulators, and environments have been proposed to support testing learning
- They can help students gain **practical experience** and learn testing techniques in an interesting way

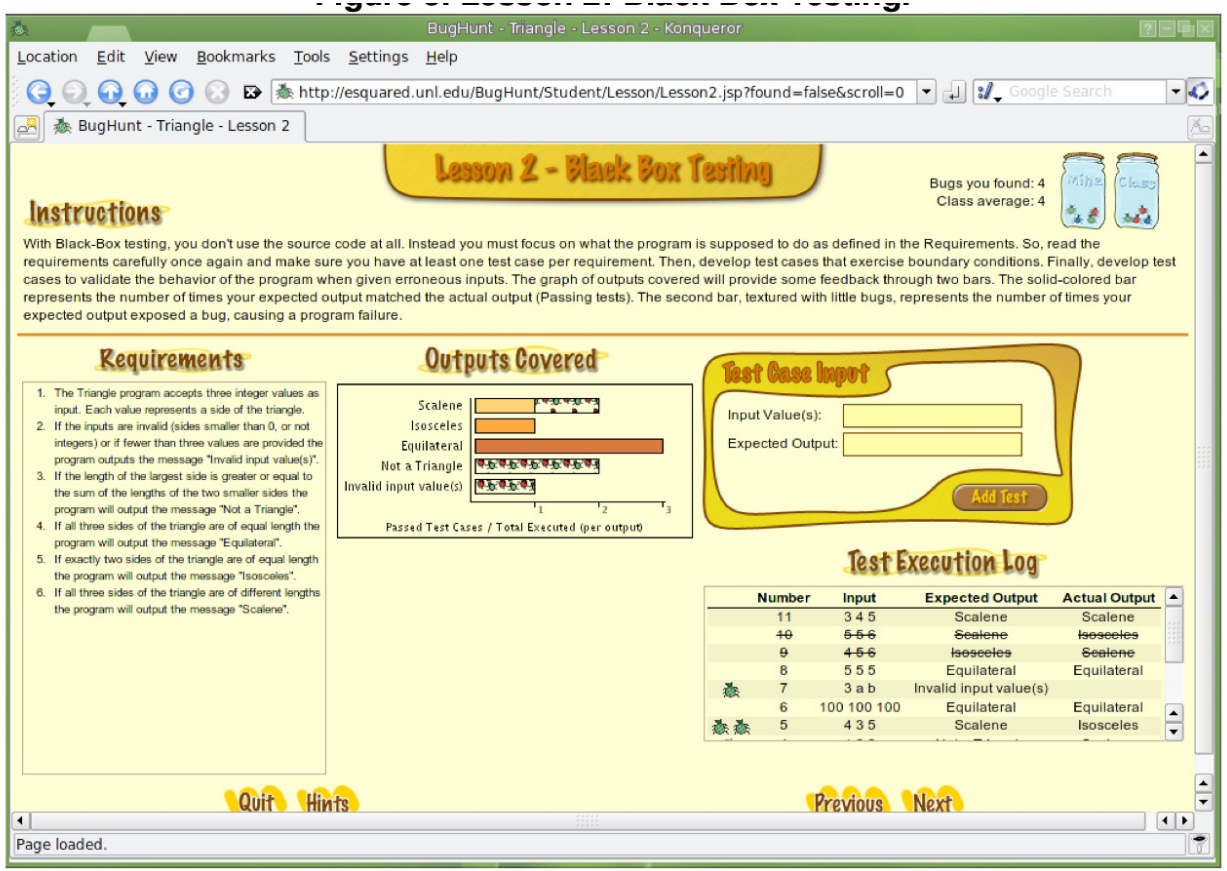
- **Three main categories** of tools emerge from the literature:
  - Tools that provide a **learning environment** to guide students through a learning path, interactively, or with the support of machine learning techniques;
  - Tools that support students in **practicing testing** in **laboratory** settings;
  - Tools that exploit a **gameful approach** to motivate students to practice testing

Learning Environments- examples

- A web-based environment dedicated to software testing (**Bug-Hunt-** 2007)
- A cyber-learning environment (**WReSTT -**2014)
- A Web IDE that can guide the students through a pre-defined path of steps to force them to write tests and specifications (**Web-IDE-** 2022)
- A web-based assignment submission platform that supports different levels of testing pedagogy via a customizable feedback engine (code coverage feedback / inquiry-based learning conceptual feedback) (**Testing Tutor- 2020)**
- 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 (**2017**)
- …

Bug Hunt – Web based environment

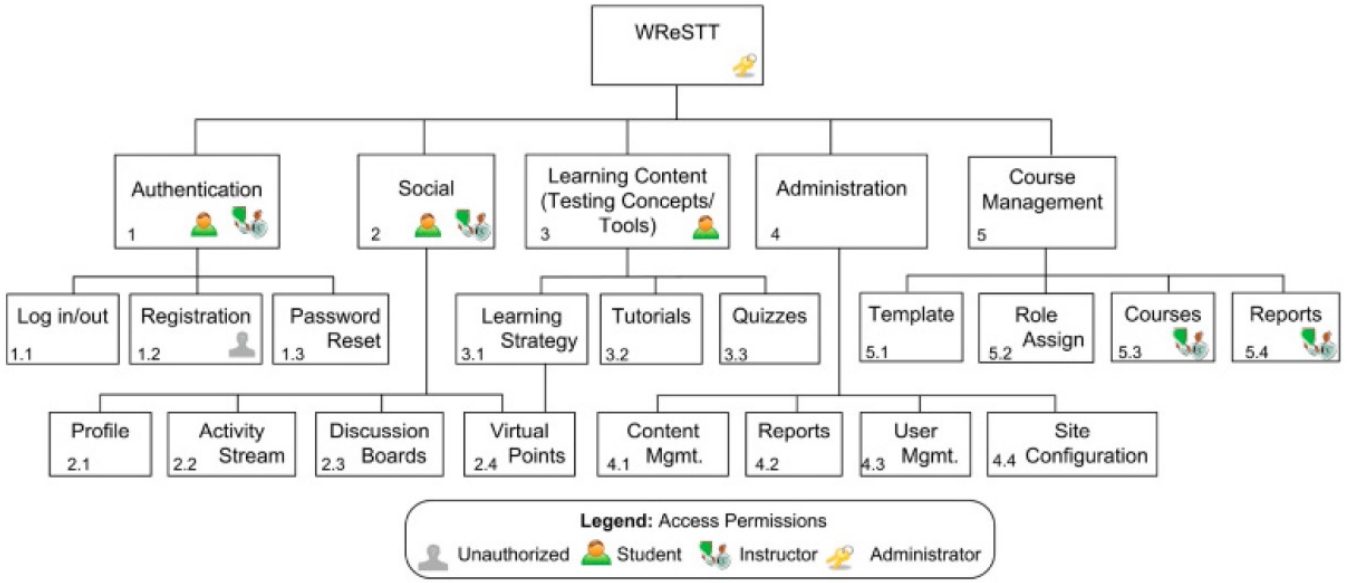
- It contains **four introductory lessons** on testing terminology, black and white box techniques, and testing automation and efficiency.
- It incorporates **challenges** in each lesson and provides immediate **feedback** to promote engagement while students practice the application of fundamental testing techniques
- It provides a complete and **automatic assessment** of students' performance to reduce the instructor's load.



*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**). IEEE, 688–697.*

WReSTT-Cyle- Web-Based Repository of Software Testing Tutorials: A Cyberlearning Environment

- Initailly started as **a Web-Based Repository** of Software Testing Tools
- It evolved in a **learning platform** based on three key learning strategies:
- **collaborative learning, problem-based learning (PBL), and gamification**.
- The platform encourages the students' engagement, their collaboration and teamwork, but also competitiveness
- It is now called **SEP-CyLE** (Software Engineering and Programming Cyberlearning Environment).



Peter J Clarke, Debra Davis, Tariq M King, Jairo Pava, and Edward L Jones. 2014. Integrating testing into software engineering courses supported by a collaborative learning environment. ***ACM Transactions on Computing Education*** *(TOCE) 14, 3 (2014), 1–33.*

An Automated System for Interactively Learning Software Testing

- 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

- The system also gives clear **examples of buggy implementations** that their tests do not detect.

Advantages

- Students are not testing their own code
- The submitted tests are evaluated based on the number of buggy programs they detect from a large corpus of buggy programs, not their code coverage
- This feedback also gives them practice reading, understanding, and debugging code written by others

Rebecca Smith, Terry Tang, Joe Warren, and Scott Rixner. 2017. An automated system for interactively learning software testing. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education. 98–103.

Tools for practicing testing in laboratory settings

- Tools providing a laboratory infrastructure that help students to practice different types of testing
  - Architecture proposal for a software testing **laboratory based on a cloud** platform (*W. Wen, et al., "Design and Implementation of Software Test Laboratory Based on Cloud Platform," 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion)*
  - An automated **framework** for unit and integration testing and grading for intermediate-level systems course projects. (*Dee A. B. Weikle, et al. 2019. Automating Systems Course Unit and Integration Testing: Experience Report. In Proc. of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19*).

Tools that exploit Gamification

- **Gamification** has been defined as «the use of game design elements in non-game contexts»
- **Gamification** may increase motivation for students studying software testing.

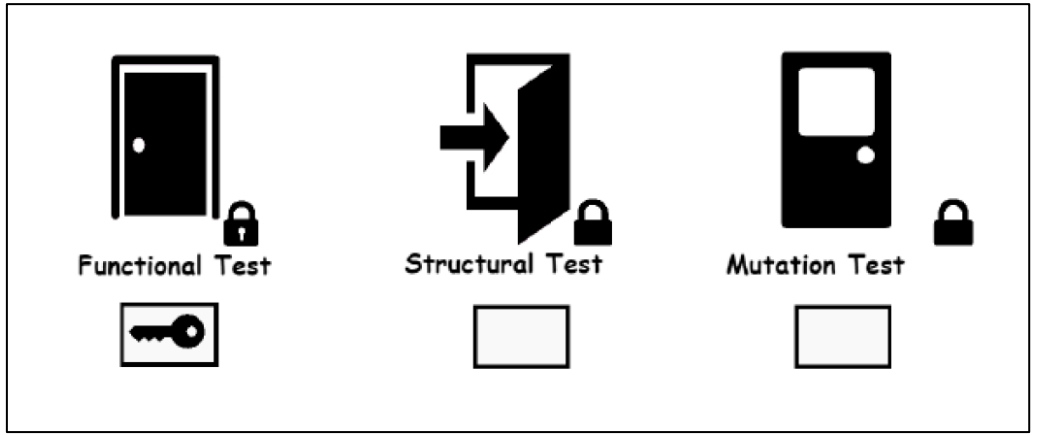
- Two main categories of tools:
  - Tools that **implement a game** with a **specific game dynamic** that try to stimulate students by challenges and competitions. They include the typical gamification elements such as reward points, badges, avatars, and leader boards in a learning environment.
  - Tools that rather offer a **simulation/ education game** where students can practice testing on small/ toy examples.

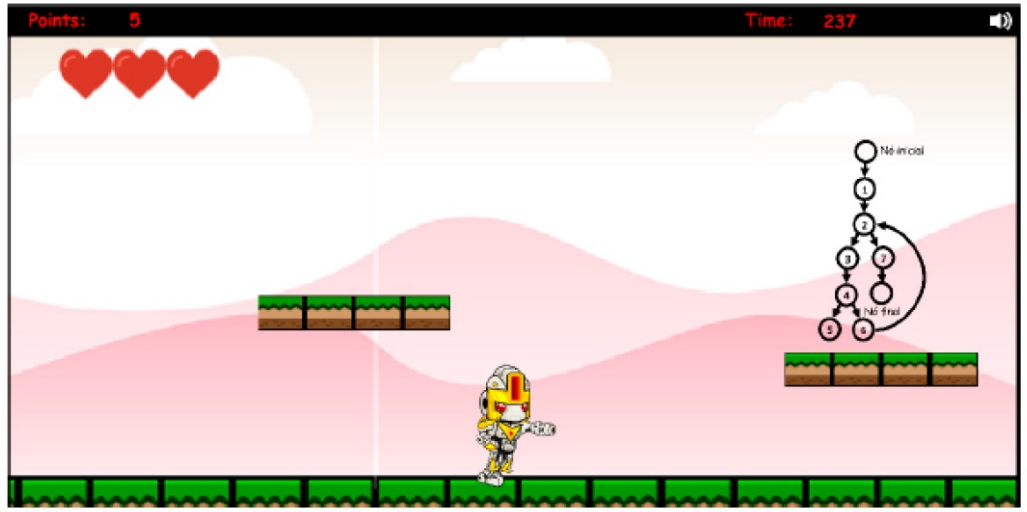
Some examples

- **HALO**: A gameful approach to teaching software design and software testing – by assignments, quests and rewards (2011)
- **Bug Catcher**: A system for software testing competitions on bug catching - provides students with requirements, buggy code, and input fields to enter test cases (2013)
- **A testing Game-** to do practice of black box, white box and mutation testing using avatars, game levels, rewards (2017)
- **Code Defenders**: a mutation testing game where players take the role of an attacker, who aims to create the most subtle non-equivalent mutants, or a defender, who aims to create strong tests to kill these mutants (since 2016)
- A gamification tool used as a systematic strategy in the teaching of **Exploratory Testing** (2019)
- An experimental **card game** for software testing (2016)
- ….

A Testing Game

- A game implemented as a Web application to do practice of black box, white box and mutation testing.
- The participant is represented in the form of an **avatar** and has a set of **abilities** similar to traditional 2D games.
- The avatar can walk, run, jump, evade or fight enemies.
- The **levels** in the game are directly mapped to the three major topics covered in game (black box, white box and mutation), and they are represented with a doors





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). IEEE, 1–8.*

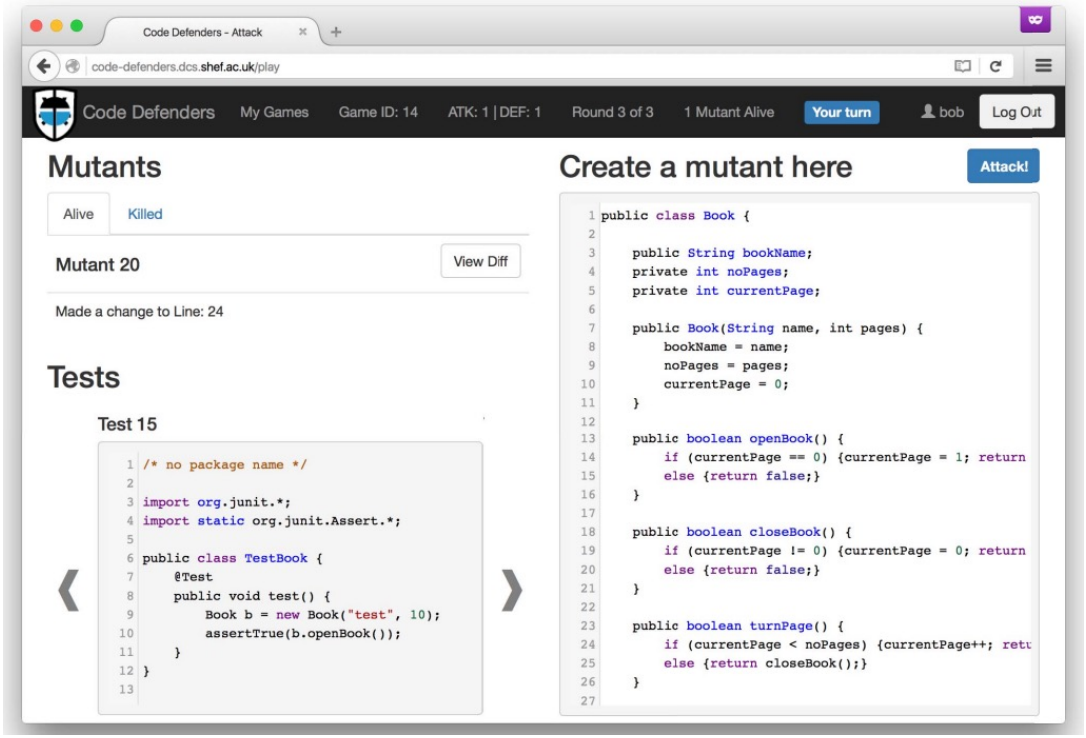
Code Defenders

- The Code Defenders game created an environment where the students can compete against each other over the code that is being under test by:
  - **attacking** (introducing bugs) or
  - **defending** (creating test cases that will expose those bugs),
  - learning the practical **mutation testing** concepts in the process.

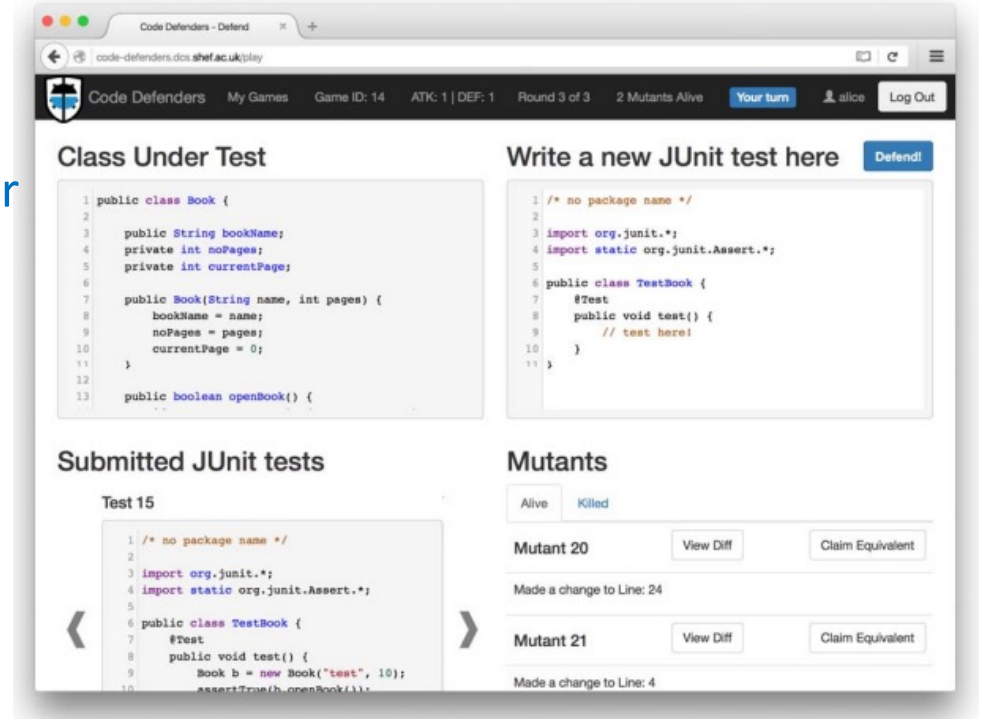
- The game can serve an educational role by engaging learners in mutation testing activities in a fun way.
- Experienced players will produce strong test suites, capable of detecting even the most subtle bugs that other players can conceive.
- Equivalent mutants are handled by making them a special part of the gameplay, where points are at stake in duels between attackers and defenders

*Gordon Fraser, Alessio Gambi, Marvin Kreis, and José Miguel Rojas. 2019. Gamifying a software testing course with code defenders. In Proceedings of the 50° **ACM Technical Symposium on Computer Science Education**. 571–577.*

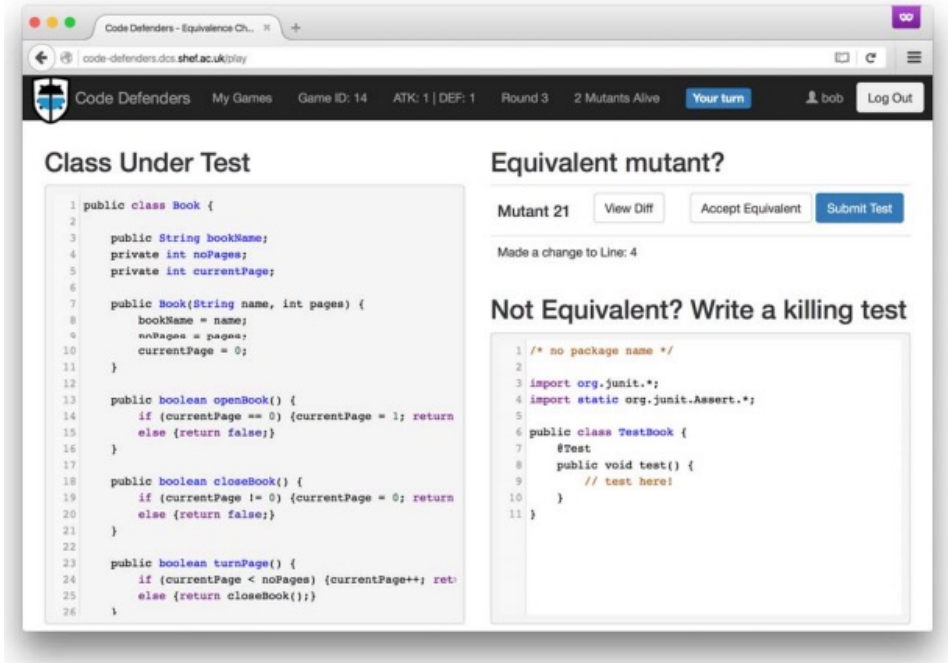
Code Defenders



The Attacker View



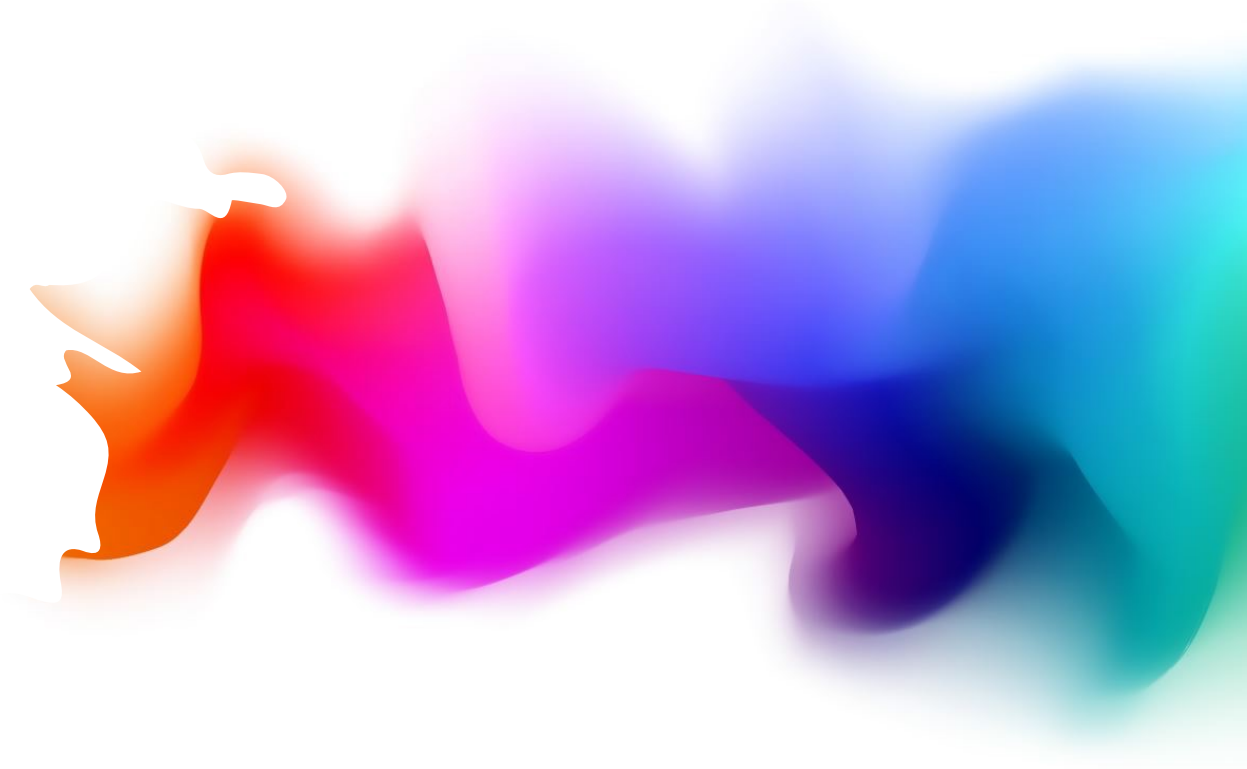
The Defender View



The Equivalent Mutant Challenge View

J. M. Rojas and G. Fraser, "Code Defenders: A Mutation Testing Game," *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Chicago, IL, USA, 2016, pp. 162-167, doi: 10.1109/ICSTW.2016.43

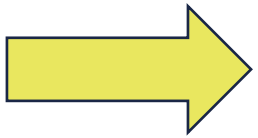
Current State of Software Testing in Education



Current state of software testing in Academic education

- Many works investigated the state of software testing in education in different parts of the world:

- ***Australia***
- ***Canada and America***
- ***Hong Kong***
- ***South Africa, Brazil, and abroad***



Emerging Results:

“More testing should be taught ”

“Results indicated a deficiency for all testing topics in practice activities.

"Negative gaps in topics such as test of web applications, functionality testing and test case generation from client requirements/user stories"

…references…

- *V. Garousi and A. Mathur, "Current state of the software testing education in north american academia and some recommendations for the new educators," in 2010 23rd IEEE Conference on Software Engineering Education and Training, 2010, pp. 89–96.*
- *P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "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), 2015, pp. 62–68.*
- *I. S. Elgrably and S. R. B. de Oliveira, "A diagnosis on software testing education in the brazilian universities," in 2021 IEEE Frontiers in Education Conference (FIE), 2021, pp. 1–8.*
- *S. M. Melo, V. X. S. Moreira, L. N. Paschoal, and S. R. S. Souza, "Testing education: A survey on a global scale," in Proceedings of the XXXIV Brazilian Symposium on Software Engineering, ser. SBES '20. ACM, 2020, p. 554–563.*

And what about teaching Testing in European academic institutions?

- A recent study (2023) investigated the context of **Sweden**…
  - *A. A. Barrett, E. P. Enoiu, and W. Afzal, "On the current state of academic software testing education in Sweden," in 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2023, pp. 397–404.*
- With the exception of this contribution, there is a lack of studies specific to the European context

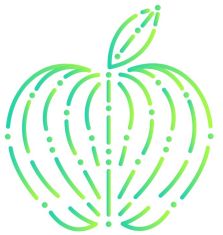
A recent investigation on the state of software testing education in four European countries

- The state of the practice in software testing education in academic institutions from ***Belgium, Italy, Portugal, and Spain*** was recently investigated in a Course Mapping study, illustrated in:

  - *State of the Practice in Software Testing Teaching in Four European Countries, P. Tramontana, B. Marin, A.C. Paiva, A. Mendez, T. E.J. Vos, D. Amalfitano, F. Cammaerts, M. Snoeck, and A. R. Fasolino, to appear in* IEEE-ICST 2024 proc.

- The study was conducted as part of the **ENACTEST Project** (https://enactest-project.eu/)
  - ENACTEST aims to investigate and improve the current practices in software testing education, by proposing new software testing teaching materials, the so-called capsules

European iNnovation AllianCe for TESting educaTion



Goal, Research Questions and Target Population

- **Goal**: To 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.

- ***RQ1****: How common are software testing related courses in the considered academic context?*
- ***RQ2****: What are the educational organisational characteristics of these courses?*
- ***RQ3****: What aspects of software testing are most commonly taught?*

- **Target Population**: Academic courses teaching software testing topics from each of the four European countries

Steps

1

University Selection

2

Course Selection

3

Data Collection

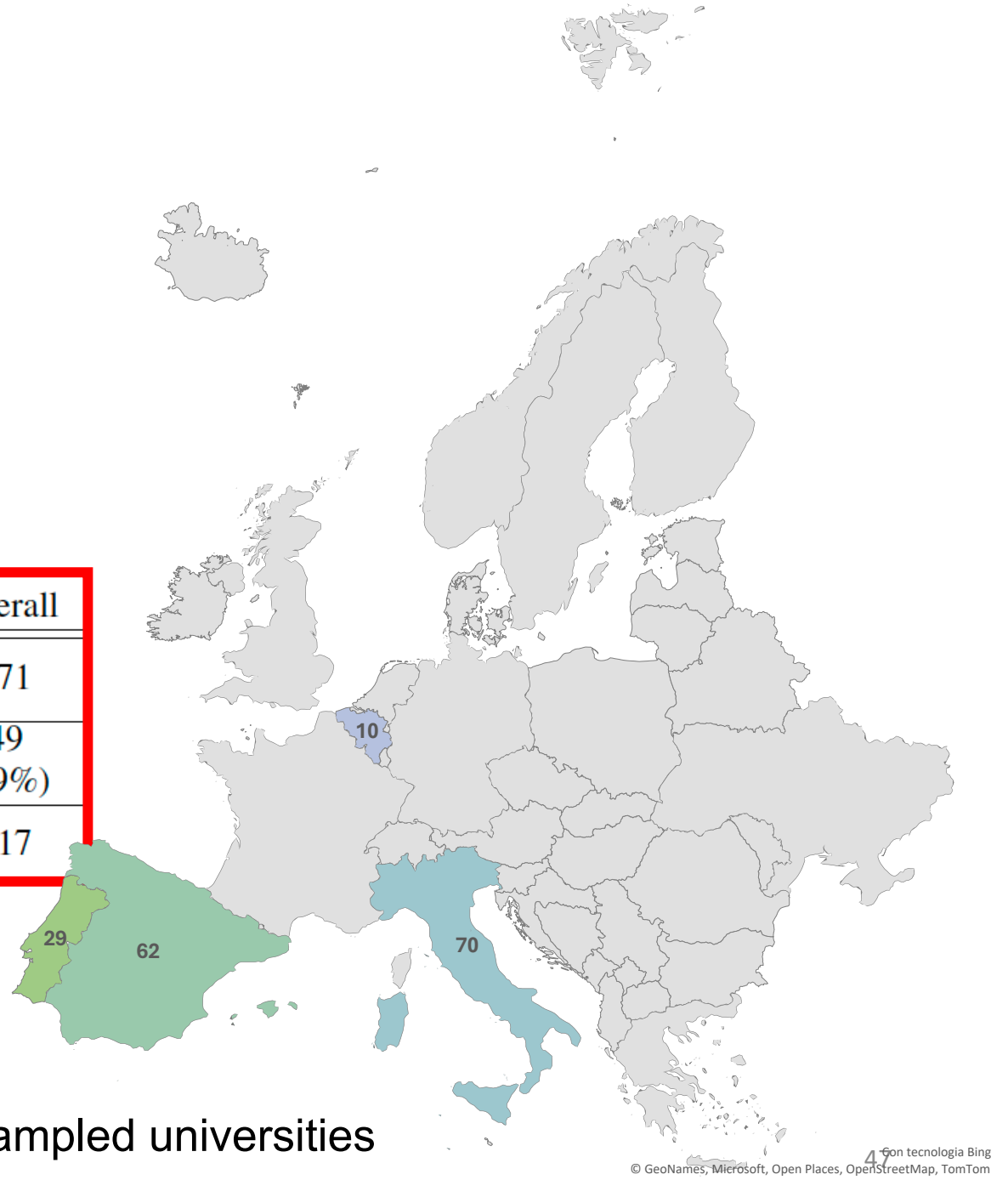
4

Data Analysis

University Selection

From the list of **171** ranked Universities from the *2023* ***Scimago*** *Institutions Ranking of Computer Science Universities…*

	Spain	Italy	Belgium	Portugal	Overall
Universities in SJR ranking	62	70	10	29	171
Randomly Selected Universities	19 (31%)	20 (29%)	3 (30%)	7 (24%)	49 (29%)
Analysed Courses	28	44	10	35	117



… We sampled at random **49** universities (approximately 30% of all the **171** listed universities).

Software Testing related courses offered by the 49 sampled universities were looked for in their institutional Web sites

Course Selection

- Bachelor's and Master's degrees related to computer science (i.e. **Computer Engineering, Computer Science, Software Engineering**, etc.) were analysed as a priority.

Relevant courses were manually selected, based on their **names**:

"*Software Testing*",
"Verification & Validation",
"Software Quality"…
"Software Engineering" and "Programming"…

Inclusion and Exclusion criteria were used to select only relevant courses:

- *The course syllabus contains a description of the course topics.*
- *The course syllabus is written in English or in the language of one of the ENACTEST Project partners*
- *The course syllabus includes testing topics.*

- ***117** courses that included Software Testing topics were admitted to the Data Collection step*

Data Collection

- From syllabus and curricula in the official websites of the academic institutions
- Extracted information:
  - Country, University Name, Degree Name, Degree Level, Course Name, Course Year, Scientific Field
  - Number of Credits, Number of Hours (Theory and Lab Hours)
  - Focus on Testing (Prevalent/ Partial),
  - Assessment Methods, P. Languages, Reference Books
  - Taught Testing Topics

Collected Data available online:
https://zenodo.org/records/10527894

We distinguished:

- **ST** courses having a prevalent focus on testing topics (i.e. more than 75% of topics correspond to testing),
- **NST** courses (non-specialized testing courses)

The extraction was done by at least two researchers per country and validated by other two researchers

Results- RQ1

- *RQ1: How common are software testing related courses in the considered academic context?*

	#Universities	#ST Courses	#NST Courses
Spain	19	7	21
Italy	20	5	39
Portugal	7	9	26
Belgium	3	1	9
Total	49	22	95

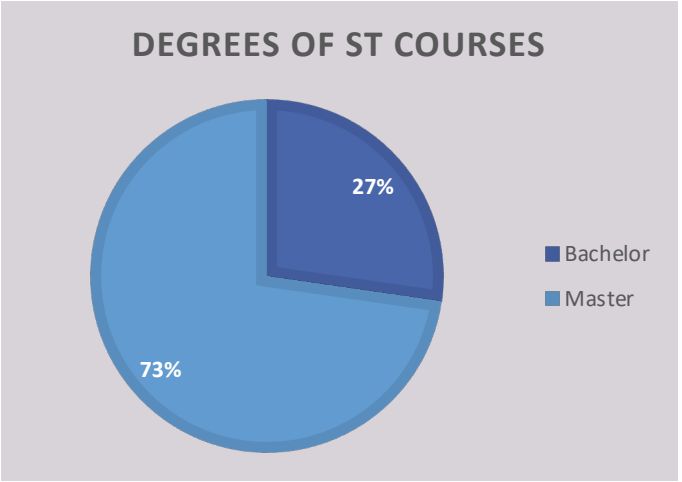
- *ST courses in 39% (19/49) of universities*
- *NST courses in 94% (46/49) of universities*
- *both ST and NST courses were offered in only 14 out of 49 universities (29%)*

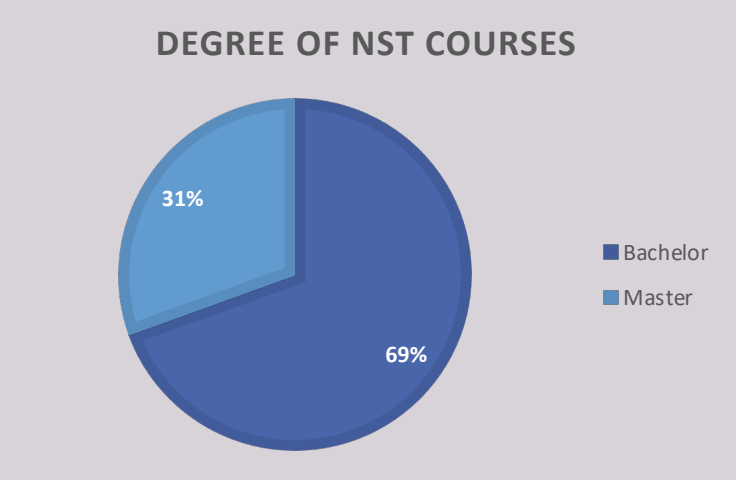
With respect to ST courses found in 56% of Swedish Universities, we found ST courses in:

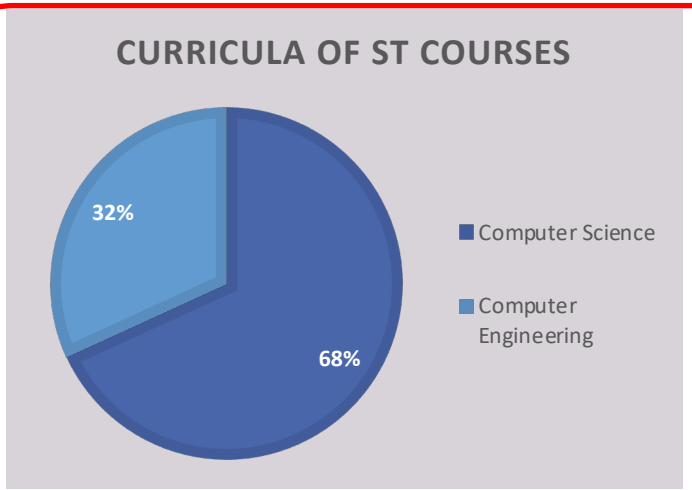
- 37% of Spanish universities,
- 33% of Belgian,
- 24% of Italian, and
- 71% of Portuguese ones.

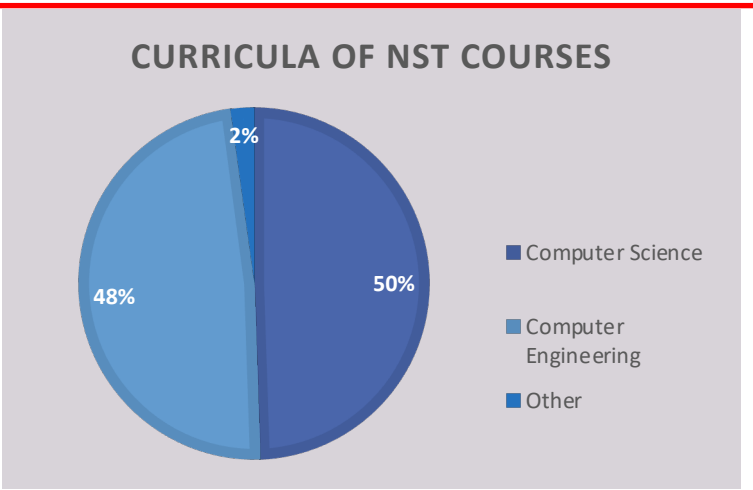
Results- RQ2

- *RQ2: What are the educational organisational properties of the courses (Degree and Curriculum)?*



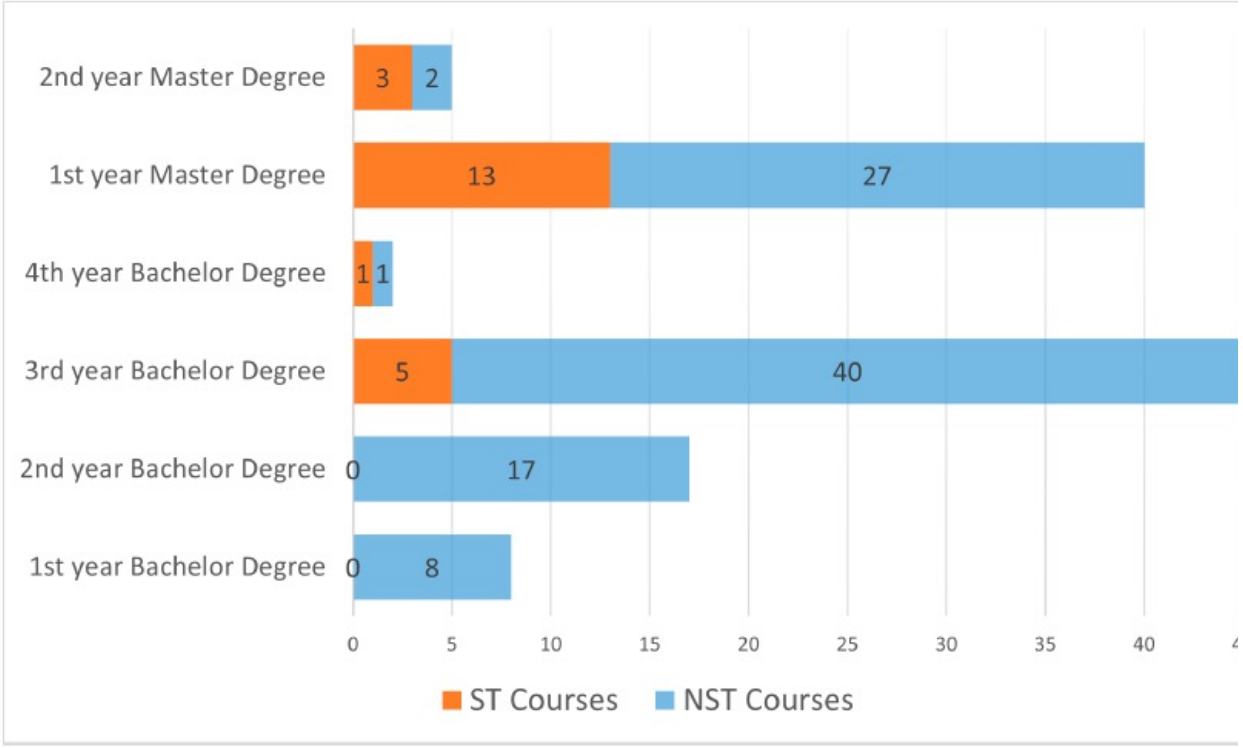






Results -RQ2: Year of Courses

- *RQ2: What are the educational organisational properties of the courses (Year)?*



Results RQ2: Course Names

ST course names -frequent terms :	ST less typical terms:	NST names:
• Software Testing (10) • Verification and Validation (8) • Software Quality (5)	• Analysis of Software Artifacts • Planning and Testing of Software Systems • Robust Software • Advanced Programming Techniques	• Software Engineering (44) • Programming (20) • Software Architecture Design (10) • Security (9) • Software Quality (7) • Software Project Management (6) • Agile Software Development (1)

Results : Course European Credits- EC

ECTS- European Credit Transfer and Accumulation System (ECTS)	ST course credits:	NST course credits:	Theoretical, practical, and laboratory number of hours:
• ECTS is a standard system for measuring the effort required by a student for a course • One EC is equal to **28** hours of study • **60 EC** credits are the equivalent of a full year of study or work.	• **6 EC** for the majority of ST courses (**19**) • **5.68** credits on **Average** for ST courses	• EC values ranging **from 3 to 12 EC**, • An average of **6.75** credits. • But it was not possible to determine exactly how much of the course time was spent on testing	• information rarely provided by the course websites, or • no standard way to describe or distinguish between them.

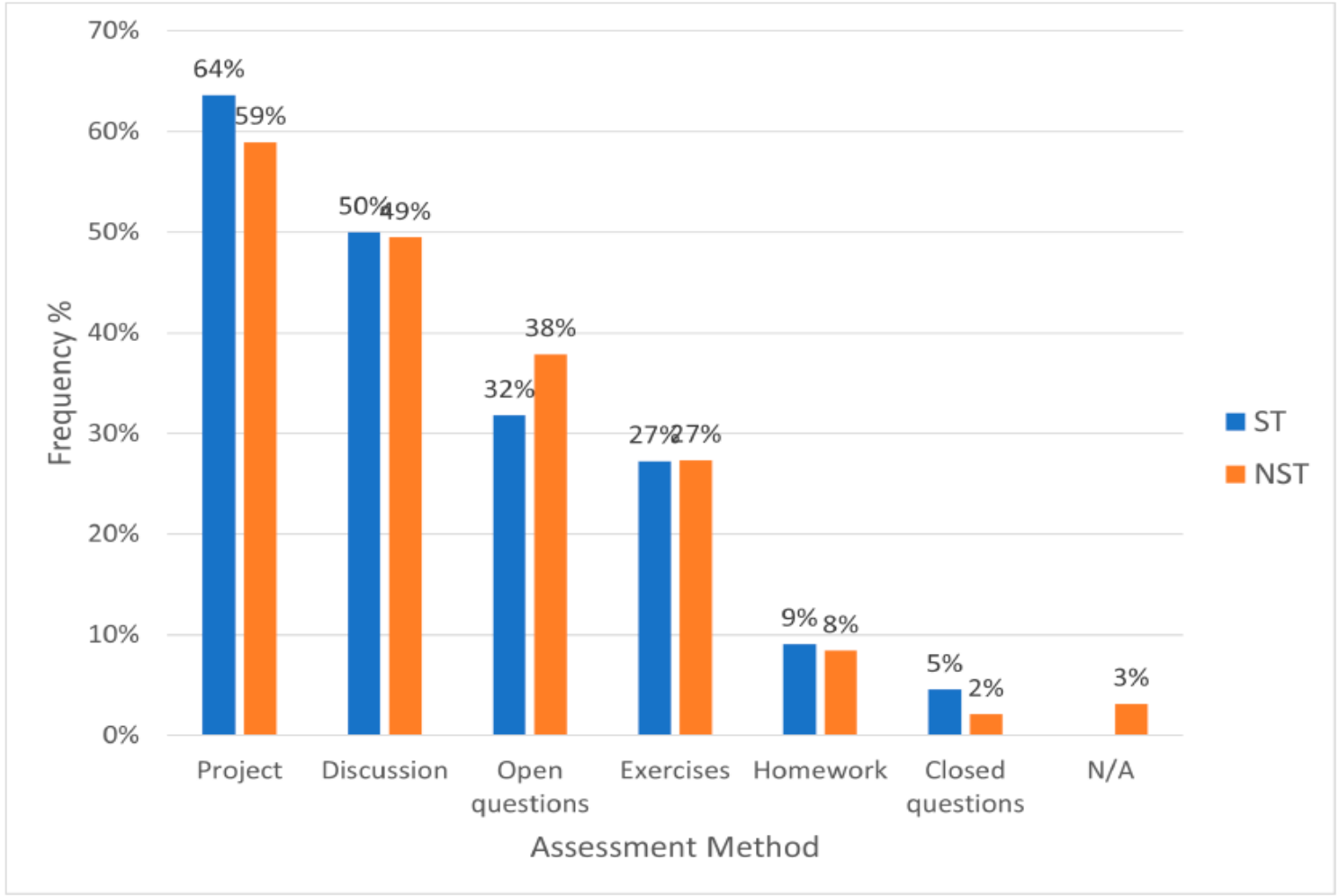
Results : Course Assessment Methods

Six assessment Methods

Open questions
Closed questions
Exercises
Homework
Project
Discussion

Most of the mapped courses adopt more than one assessment method

ST and NST courses present similar percentages for each method



Results

- *RQ3: What aspects of software testing are commonly taught?*

Testing Topics classified according to the Conceptual framework for testing offered by the **ISO/IEC/IEEE 29119 standard on Software Testing**.

Test Design Technique
Testing Practice
Testing Level
Testing Type

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		

Results- Test Design Techniques

- Specification-Based and Structure-Based techniques:
  - present in 95% and 100% of ST courses
- Experience-Based:
  - Only in 9% of ST courses
- Other:
  - In 18% of courses mutation-based testing was taught (4 ST courses)
- NST courses:
  - Specification-Based (37%)
  - Structure-Based (31%)

	ST	%	NST	%
Specification Based	21	95%	35	37%
Equivalence Partitioning	11	50%	1	1%
Classification tree method	1	5%	0	0%
Boundary value analysis	6	27%	2	2%
Syntax testing	3	14%	0	0%
Combinatorial testing	6	27%	2	2%
Decision table testing	6	27%	0	0%
Cause-effect graphing	2	9%	0	0%
State transition testing	8	36%	0	0%
Scenario testing	1	5%	0	0%
Use case testing	1	5%	0	0%
Random testing	4	18%	0	0%
Metamorphic testing	2	9%	0	0%
Requirements-based testing	1	5%	3	3%
Structure Based	22	100%	29	31%
Statement testing	5	23%	9	9%
Branch testing	5	23%	6	6%
Decision testing	4	18%	7	7%
Branch condition testing	4	18%	2	2%
Branch condition combination testing	4	18%	0	0%
MC/DC testing	3	14%	1	1%
Data flow testing	5	23%	0	0%
Experience Based	2	9%	0	0%
Error guessing	0	0%	0	0%
Other	4	18%	12	13%

Results- Testing Practices

- Most common Practices in ST:
  - Automated Testing
  - Model-Based testing
  - And then Scripted, Exploratory, Manual
- In NST courses:
  - Automated testing
  - Scripted
  - Manual
- Other practices:
  - Regression Testing
  - TDD

Testing Practices	ST	%	NST	%
Model-based testing	7	32%	5	5%
Scripted testing	3	14%	6	6%
Exploratory testing	3	14%	1	1%
Experience-based testing	1	5%	0	0%
Manual testing	3	14%	6	6%
A/B testing	0	0%	4	4%
Back-to-back testing	0	0%	0	0%
Mathematical-based testing	3	14%	0	0%
Fuzz testing	2	9%	0	0%
Keyword-driven testing	0	0%	0	0%
Automated testing	15	68%	20	21%
Other	6	27%	26	27%

Results- Testing Levels

- Most common Levels in ST:
  - Unit, Integration, System (similar)
- In NST courses:
  - Unit Testing
- Acceptance Testing
  - Less Frequent in NST (only 13%)

Testing Levels	ST	%	NST	%
Unit testing	13	59%	48	51%
Integration testing	10	45%	22	23%
System testing	10	45%	18	19%
System integration testing	4	18%	0	0%
Acceptance testing	7	32%	12	13%
Other	4	18%	12	13%

Results- Testing Types

- Most common Types in ST and NST:
  - Functional Testing
  - Performance
  - Security
- Many other types not mentioned at all

Testing Types	ST	%	NST	%
Functional testing	8	36%	40	42%
Accessibility testing	0	0%	1	1%
Compatibility testing	0	0%	0	0%
Conversion testing	0	0%	0	0%
Disaster recover testing	0	0%	0	0%
Installability testing	0	0%	0	0%
Interoperability testing	0	0%	0	0%
Localization testing	0	0%	0	0%
Maintainability testing	0	0%	0	0%
Performance related testing	4	18%	6	6%
Portability testing	0	0%	0	0%
Procedure testing	0	0%	0	0%
Reliability testing	0	0%	2	2%
Security testing	2	9%	9	9%
Usability testing	1	5%	1	1%
Other	5	23%	9	9%

Main Findings- diffusion of ST and NST courses

- Software Testing dedicated courses were found in **39%** of the 49 universities surveyed (mostly offered at Master degree level),
  - *Ardic and Zaidman found software testing dedicated courses in **50%** of the top 100 of the Times Higher Education university ranking*
  - *Barrett et al. found **56%** of universities with dedicated ST courses in the Swedish context*

- Software testing fundamentals were included in at least one course at every university (i.e., in 94% of surveyed universities)
- Most of the existing advanced software testing techniques are not taught to students and future IT professionals.
  - *Far Too Limited!!!*

Main Findings- theoretical vs practical classes

- Difficult to find data about the number of hours of theoretical classes and practical classes for each mapped course
- However, in 59% of ST courses we found similar amount of hours for theoretical and practical classes, with a ratio of 1.03 between them

✓ *This approach is both a more challenging and attractive strategy for students!*

- Assessment Methods mostly based on Projects (in 60% of courses)

✓ *This is aligned with the practical experience needed to learn complex topics!*

- A lot of fragmentation regarding Software Testing Teaching Books

Main Findings- Testing Topics

- Structure Based and Specification Based Testing almost always present, but..

✓*Experience Based and Exploratory testing almost always neglected in the analysed 117 courses!*

- Testing courses more inclined towards the **analytical school**, *where the emphasis is on better testing through improved precision of specifications*, instead of the **context-driven school** that emphasizes exploratory testing

- Some Testing Types systematically neglected by the analysed courses:
  - accessibility testing, security testing, and disaster recovery testing, among the others

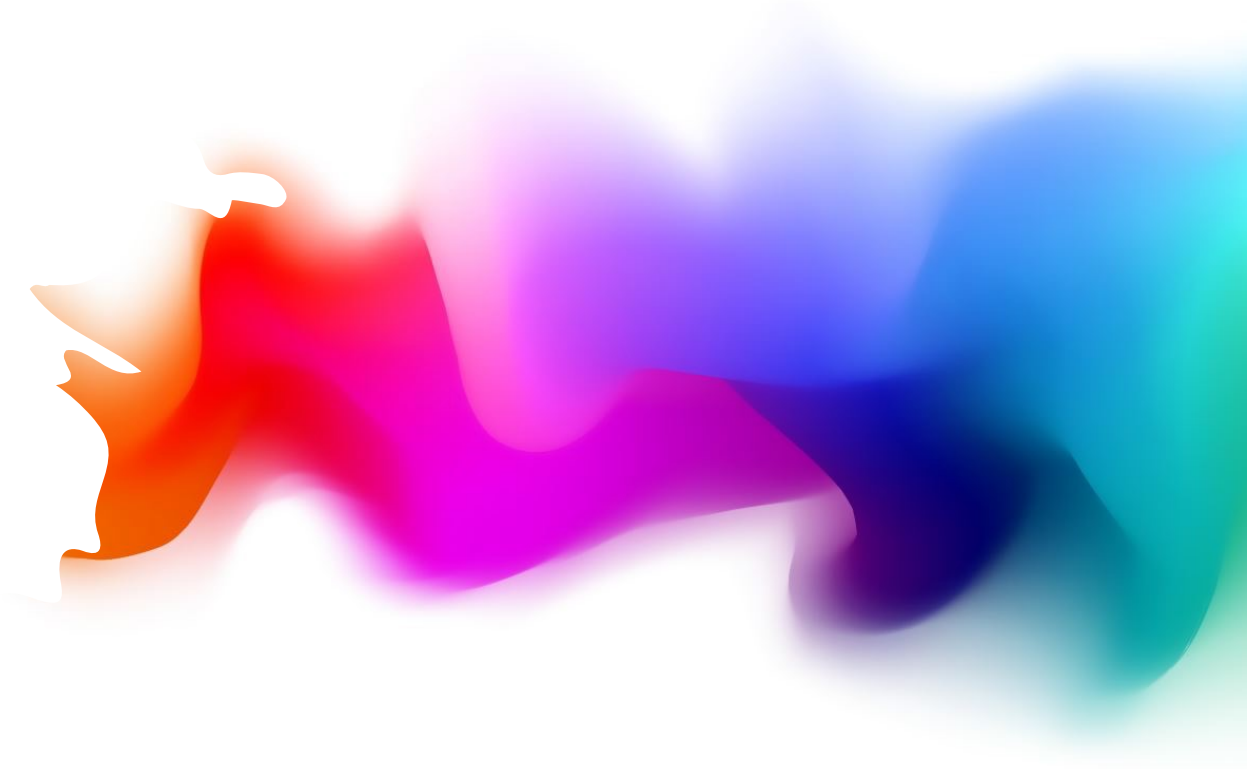
✓*There is the necessity to bring current academic offerings closer to the real needs of the industry!*

Other findings

- More standardized ways to describe the academic characteristics of courses should be systematically adopted to facilitate the course analysis
- A consolidated repository of academic courses in Europe is lacking

- Threats to validity
  - Construct (risk of misinterpretation of collected information, due to non standard descriptions on the web sites)
  - Internal (only publicly available information was processed, but having different levels of detail)
  - External (The courses we selected may not be generalized to other countries, or may not reflect the current general landscape of software testing education in Europe at large)

A Closer look at the Teachers' point of View



What is the teachers' perspective?

- The Course Mapping study offers a high-level view about the status of software testing education in the 4 European countries
- But it does not give specific information about:

  - What is the detailed organization of the course?
  - What Testing teaching practices do teachers use in their course?
  - What Teaching Learning Tools do they adopt?
  - What Gamification approaches do they propose?
  - Are the teachers satisfied about the available materials? What are their main desiderata?

A teacher survey to fill-in this gap…

- We designed and distributed a survey to Software Testing teachers in order to collect their perspectives about the considered topics

- At the moment the survey has been distributed to teachers from several European countries

- It is still possible to participate in the survey
  - if you are a software testing teacher in an academic/ research institution and you are interested, please contact me by email ….

European iNnovation AllianCe for TESting educaTion

Future Perspectives in Software Testing education

Keeping curriculum up-to-date

- Software testing methodologies, tools, and technologies evolve rapidly.
- Educators must ensure that their curriculum reflects these changes to equip students with the most relevant skills for the industry.

Focus on automation and DevOps

- With widespread adoption of DevOps practices and Continuous Integration/Continuous Deployment (CI/CD), there is a growing demand for skills in automated testing and continuous integration.
- Educators must prepare students with the necessary skills to work in DevOps environments and understand how to integrate testing throughout the software development lifecycle.

Non-functional testing aspects

- Beyond functional testing, there is an increasing importance placed on non-functional testing aspects such as security, performance, accessibility, and usability.
- Educators need to provide students with a deep understanding of these concepts and related testing techniques.

Soft skills and collaboration

- Software testing involves not only technical skills but also soft skills such as teamwork, effective communication, and problem-solving.
- Educators must integrate hands-on learning opportunities and collaborative projects into their curriculum to develop these skills in their students.

Enhance engagement, motivation, and learning

- Traditional teaching methods struggle to engage students effectively, especially in software testing.
- Gamification adds elements of fun, competition, and interactivity, which can significantly increase student engagement and motivation to learn.
- Challenge-based learning initiatives (like Hackatons and Competitions) can guide students in a learning path through exploration, collaboration, and problem-solving processes

Integration of new technologies

- Emerging technologies such as artificial intelligence, Generative AI, Internet of Things (IoT), and cloud computing require the integration of new testing concepts and tools into educational programs.
- Educators need to find ways to incorporate these technologies into their lessons and practical labs.
- Open Repositories offering different types of «Testing Learning Objects» to the educator community are needed

The experience of the ENACTEST project

(https://enactest-project.eu/)

- In ENACTEST we are investigating learning strategies and designing seamless teaching materials for testing that are aligned with industry and learning needs.

Online Game for for the early introduction of testing for initial programmers	Analogue Card Game for practice of testing strategies to deal with risks and considering quality attributes of SUTs.	Learning Environment for model-based testing	Learning Environment for Test Smell detection and removal learning	Game for practicing Coverage-based automated testing by challenging an automated test generator	Online game where student can create mutants and test cases on models.	Teaching and evaluation materials for state-based testing	Teaching and evaluation materials for BDD testing	Penetration testing leveraging Dynamic Application Security Testing (DAST)

Funded by the European Union

ERASMUS plus Project 2022-2025

European iNnovation AllianCe for TESting educaTion

Conclusions

Conclusive Remarks

- We have analysed open challenges in Software Testing Education and presented possible solutions, their benefits and limitations
- The State of the Practice in software testing teaching shows that there is still much work to do (in terms of course diffusion, topics to be taught, practical approaches to introduce, …)
- Different types of initiatives (alliances among educators, teachers and industry, competitions, gamification, sharing experiences, teaching materials…) are needed to make progress in this important field


Thank you for the attention!