Pragmatic software testing education

Aniche, Maurício; Hermans, Felienne; van Deursen, Arie

DOI
10.1145/3287324.3287461

Publication date
2019

Document Version
Other version

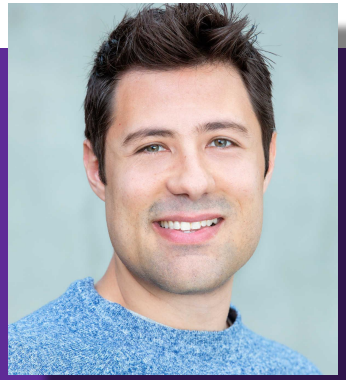
Published in
SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education

Pragmatic Software Testing Education

(experience report)

Maurício Aniche, Felienne Hermans, Arie van Deursen

TU Delft



Why software testing?

- **Because society needs software that works!**
- Fundamental activity in software development.

- **Academic perspective:** several open challenges.
- **Industry perspective:** "Very popular" nowadays, companies want developers who know how to test.

Teach software testing can be tricky

- ST is an elective course in many universities (Wong, 2012)
- Little attention is given to ST, given the large number of topics already covered (Clarke et al., 2014)
- Lack of educational tools and integration with other courses.
- **A clear curriculum (topic of today)**

Worlds Apart

- Developers and academia talk about different things when it comes to software testing.
- Example: the term *"automated software testing"*.

FOCUS: SOFTWARE TESTING

Worlds Apart

Industrial and Academic Focus Areas in Software Testing

Vahid Garousi, Hacettepe University

Michael Felderer, University of Innsbruck

// A comparison of the titles of presentations in several industrial and academic conferences on software testing revealed different focus areas of industry and academia. This situation seems to be one reason for low industry–academia collaboration in software testing.//

practitioners working in [insert any SE subarea here]." However, honestly, many conferences fail to really achieve that. Certain conferences have had some success—for example, the industry tracks of the International Conference on Software Engineering (ICSE) and International Conference on Software Testing, Verification and Validation (ICST). But much more must be done to really "bring researchers and practitioners together."

Toward that end, we focus here on software testing as a representative area of SE. To determine how industry and academia approach software testing, we compared the titles of presentations from selected conferences in each of the two communities. The results shed light on one cause of low IAC in software testing and led to suggestions on how to improve this situation. (For a look at what other researchers have done in this area, see the sidebar "Related Work in the Analysis of Software-Testing Research and Practice.")

Our Approach

Figure 1 depicts our analysis ap-

Garousi, Vahid, and Michael Felderer. "Worlds apart: industrial and academic focus areas in software testing." IEEE Software 34.5 (2017): 38-45.

Academics

- The oracle problem
- Test case generation
- Search-based software testing
- Model-based software testing

Developers

- xUnit frameworks
- The Testing Pyramid
- Mocking
- What to test? What not to test?

How to combine both perspectives?

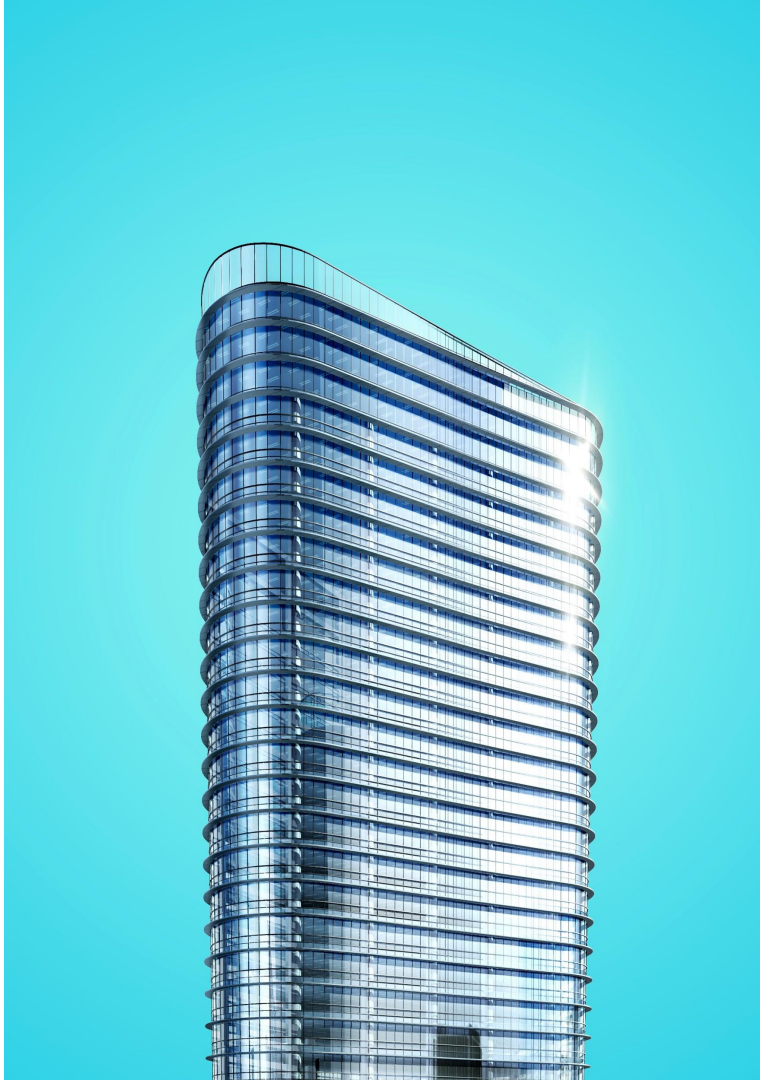
(This list was not developed in a systematic way)

Pragmatic software testing @ TU Delft

- 4th quarter, 1st year CS
  - The real main requirement is to know Java
- 5 ECTS (140 hours)
  - 2 lectures of 1.5h/week
  - 4 hours of labwork/week
- Large number of students
  - In 2017, 300 students.
  - In 2018, 450 students.
  - In 2019, ~750 students (expected)
- Assessment:
  - Midterm + exam (MC questions) 75%
  - Labwork 25%

9 key elements!

- Theory applied in the lecture.
- Real-world pragmatic discussions.
- Build a testing mindset.
- Software testing automation.
- A hands-on labwork.
- Test code quality matters.
- Design systems for testability.
- Mixture of pragmatic and theoretical books
- Interaction with practitioners.



Applied theory

- *We believe in theory*. But we prefer *applied theory*.
- Whatever theory we show to them, we also show to apply it in practice, e.g.,
  - We devise tests for a state machine, and later write them as JUnit tests.
  - We devise tests for complex decision tables, and later we write them as JUnit tests.
- Important: the theory and the practice happen during our lecture.
  - Students see us coding.

Pragmatic discussions

- Software testing is challenging, and exhaustive testing is impossible. In practice, developers have to make trade-offs.
- We discuss such real-life trade-offs:
  - How much should I test?
  - Should this piece of code be tested at unit or integration level?
  - Should I mock this class?

Building a testing mindset

- Students might not see testing as "something cool".
- We want them to have a testing mindset. How?
  - Showing that testing is a creative activity
  - Showing how to apply it in practice
  - Bringing guest lectures who discuss the importance of software testing

Software testing automation

- Industry has been long advocating the use of test automation.
- Our students:
  - Devise tests using theoretical knowledge (e.g., they devise test cases in a piece of paper)
  - They later write the tests in an automated manner.
- To that aim, students need to learn tools:
  - Junit
  - Mockito
  - Selenium / Cucumber

Hands-on labwork

- We consider getting your hands dirty highly important when learning software testing.
- Students apply all their knowledge in JPacman.
  - 3k lines of code
  - Devised specially for this course
  - Enables them to practice the different techniques we teach.
- Open source:
  https://github.com/serg-delft/jpacman-framework
  (we are still working on a Teachers' Guide. If you want to use it now, just email us)

Test code quality matters

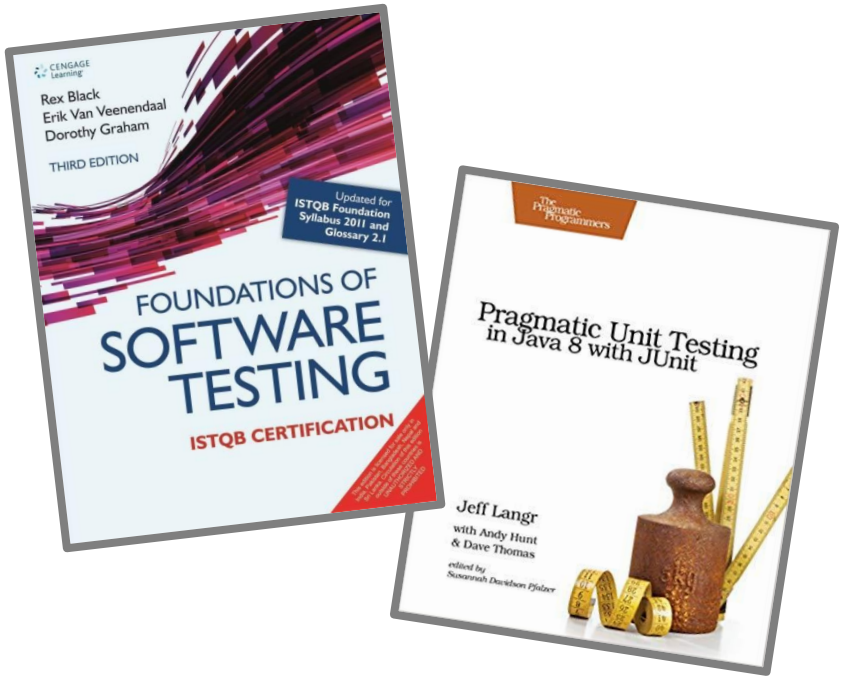
- Real life developers have to deal with large test codebases.
  - Empirical research shows that test smells are common...
- Code quality matters.
- Our TAs are trained to be picky!

Design for testability

- Software architecture meets software testing.
- More specifically, we discuss how to design systems that ease testability, e.g.,
  - How can I test my application when it depends on a database?
  - How can I test my mobile app if it depends so much on the Android's APIs, which are unmockable?
- JPacman is a good example of a testable system.

Theoretical and practical books

- We consider both the theoretical and the pragmatic part important.
- Thus, we suggest different books for these two different aspects.
  - Theory: ISTQB
  - Practice: Pragmatic Unit Testing in Java 8.



Interaction with practitioners

- Guest lectures are fundamental for students.
- Maybe not for them to learn any new practice, but for them to see how trade-offs happen in practice.
- We have tried online AMA sessions.

We were curious...

- **RQ1:** What common mistakes do students make when learning software testing?
- **RQ2:** Which software testing topics do students ind hardest to learn?
- **RQ3:** Which teaching methods do students find most helpful?

Research methodology:

- Survey with 84 students.
- Survey with 10 TAs.
- Analysis of ~2k feedback statements from our TAs to students.

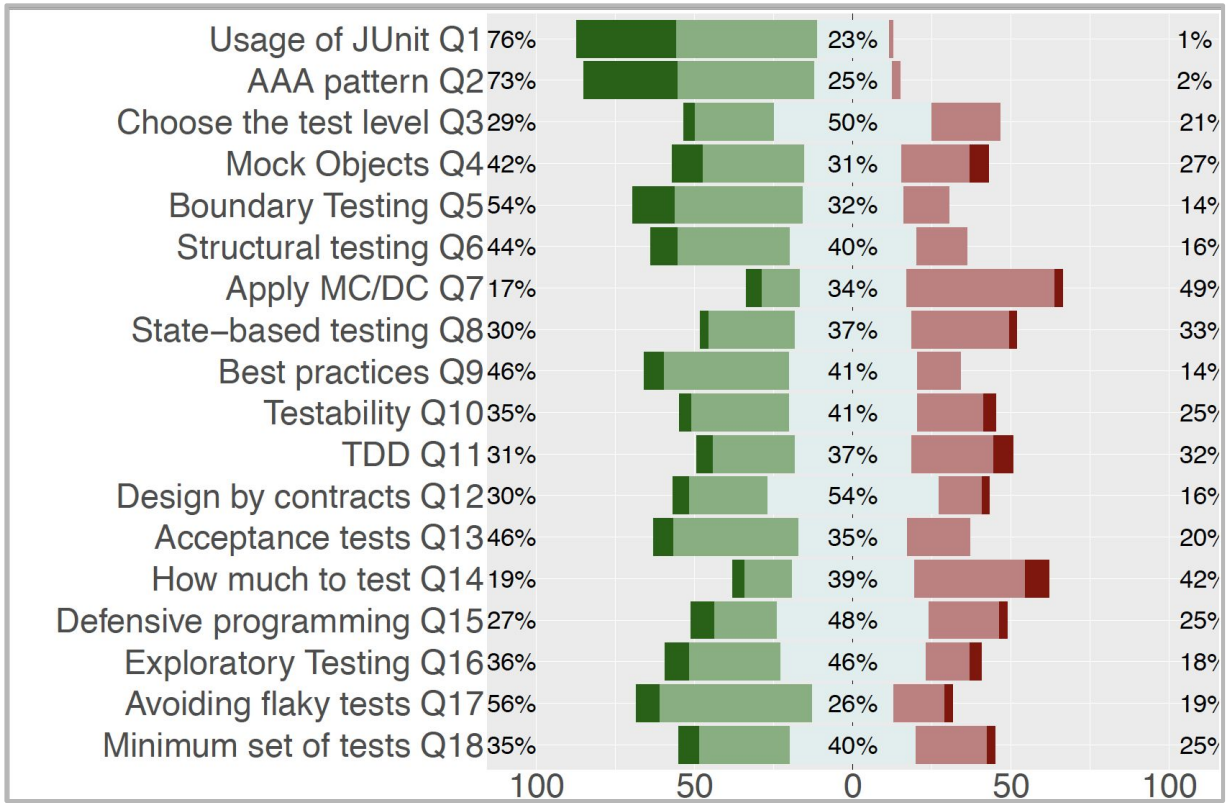
Their common mistakes

- Test coverage (416 times, 20.87%).
  - Students commonly either miss tests, i.e., they do not provide all the expected tests for a given piece of code, or they write tests that are not totally correct, e.g., the test does not actually test the piece of code.
- Maintainability of test code (407 times, 20.42%).
  - Better naming and excessive complexity, code duplication and lack of reusability, tests that could be split in two, better usage of test cleanup features, such as JUnit's Before and After.
- Understanding testing concepts (306 times, 15.35%).
  - Advantages and disadvantages of unit and system tests, and the importance of removing test smells.
- Boundary testing (258 times, 12.95%).
  - Students miss some of the boundaries.

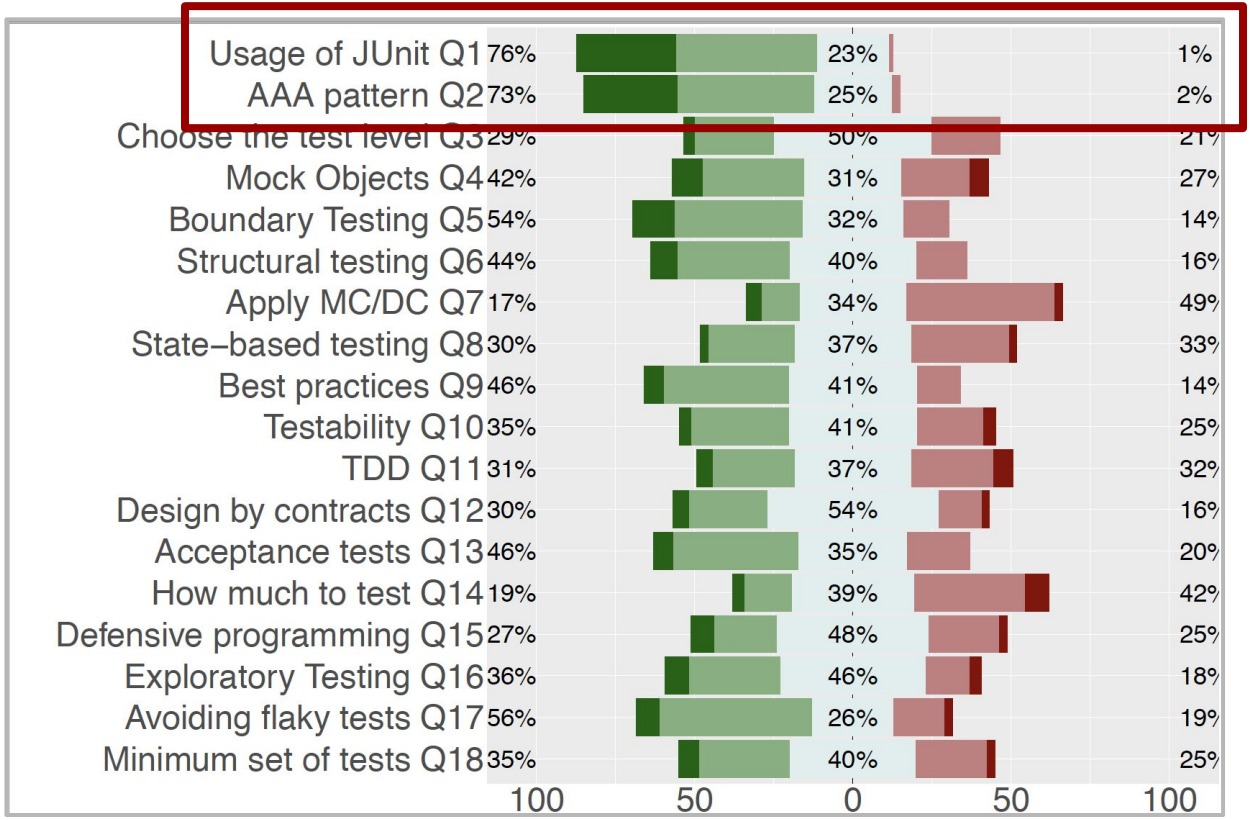
Their common mistakes

- State-based testing (247 times, 12.39%)
  - students often miss or create wrong states or events (56) and transitions (72).
- Assertions (158 times, 7.93%)
  - Missing assertions.
- Mock Objects (117 times, 5.87%)
  - how to properly verify interactions with mock objects (i.e., Mockito’s ‘verify’ method) and to explain when one should mock an object.
- Tools (84 times, 4.21%).
  - AssertJ and Cucumber can be tricky to use.

Topics hard to learn



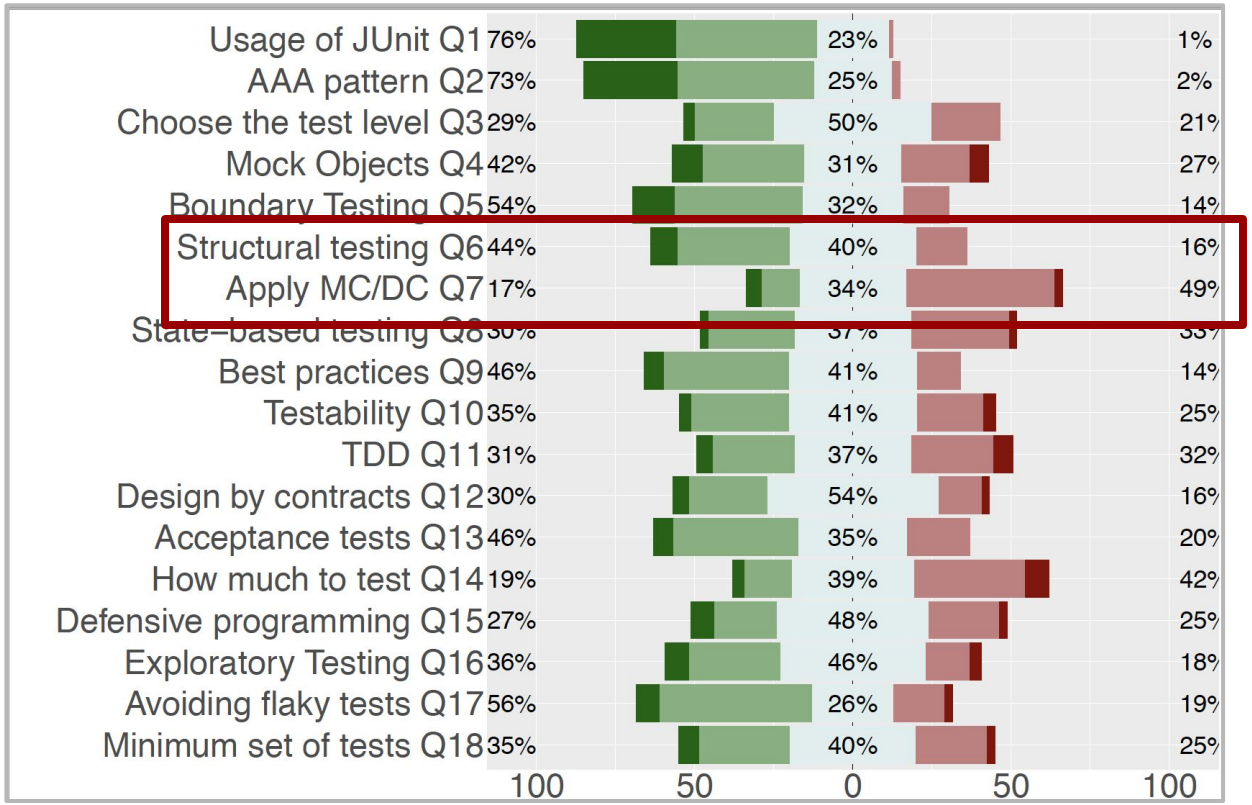
Topics hard to learn



Using the JUnit framework (Q1) as well as to think about the Act-Arrange-Assert pattern that composes any unit test (Q2) easy to learn.

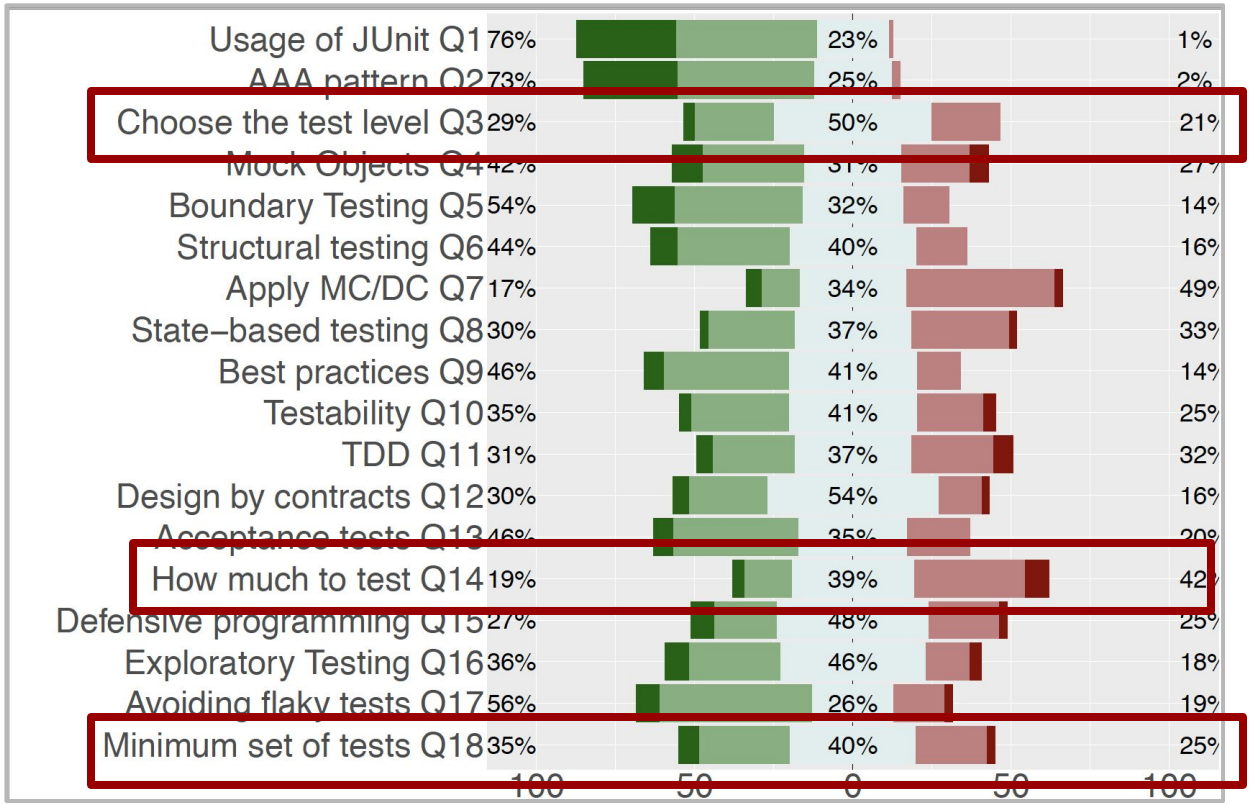
(Matches the number of feedback related to tools in previous RQ)

Topics hard to learn



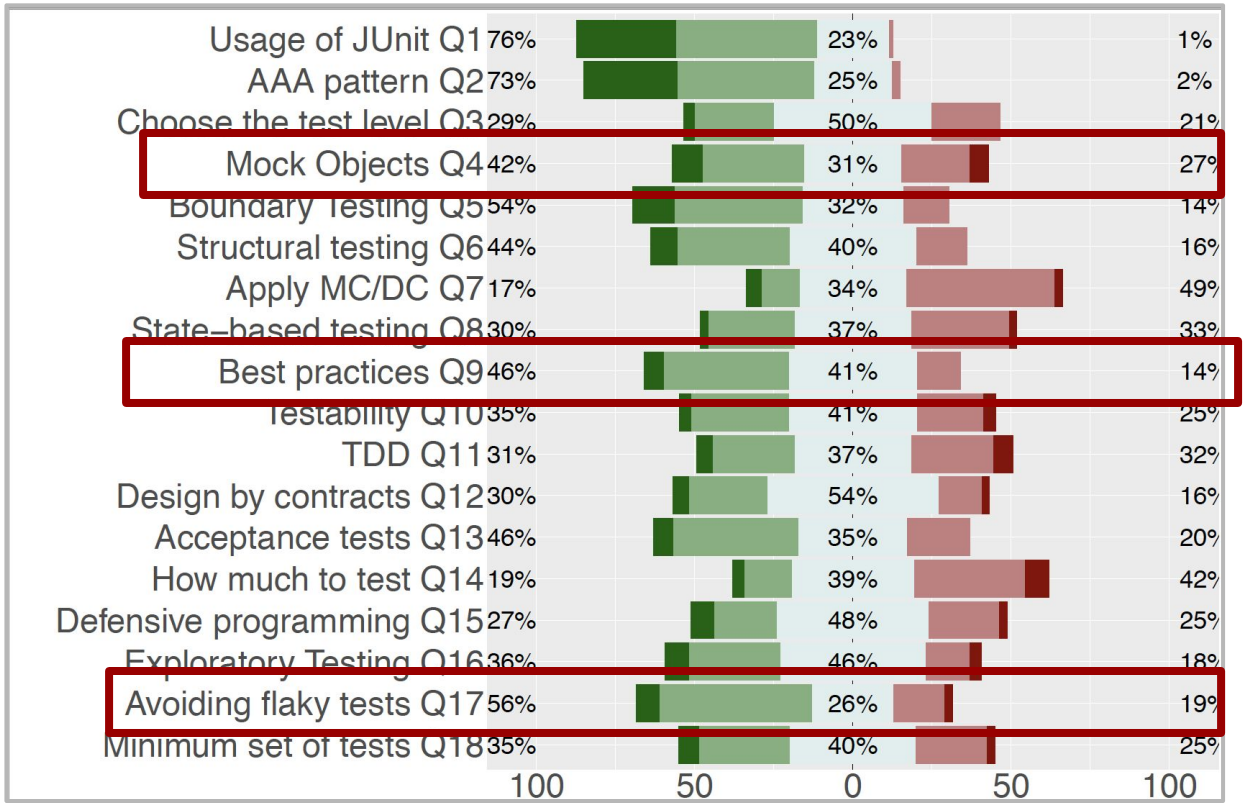
MC/DC is not an easy coverage criteria. However, structural testing in general was considered a somewhat easy topic.

Topics hard to learn



Pragmatism (choose the right test level, how much to test + minimum set of tests that gives confidence) is not easy to learn.

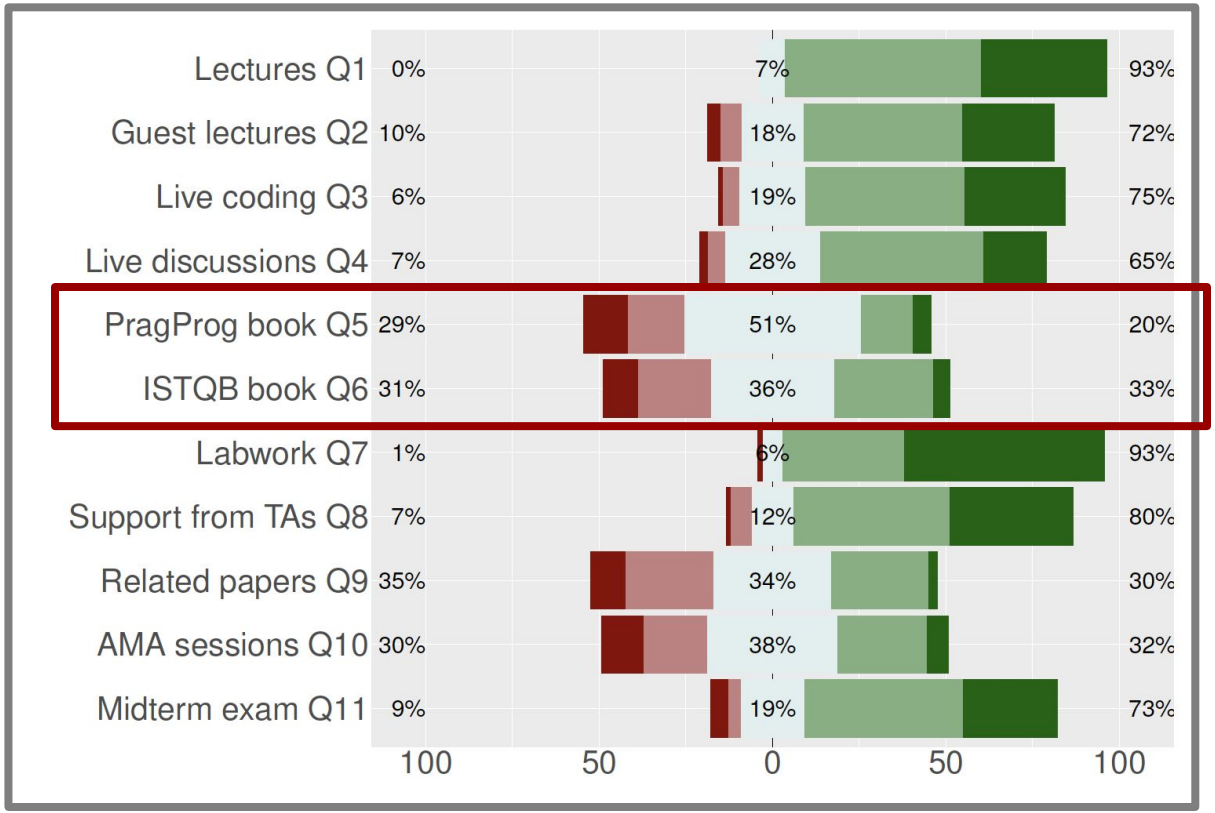
Topics hard to learn



Students think Mock Objects are an easy topic.

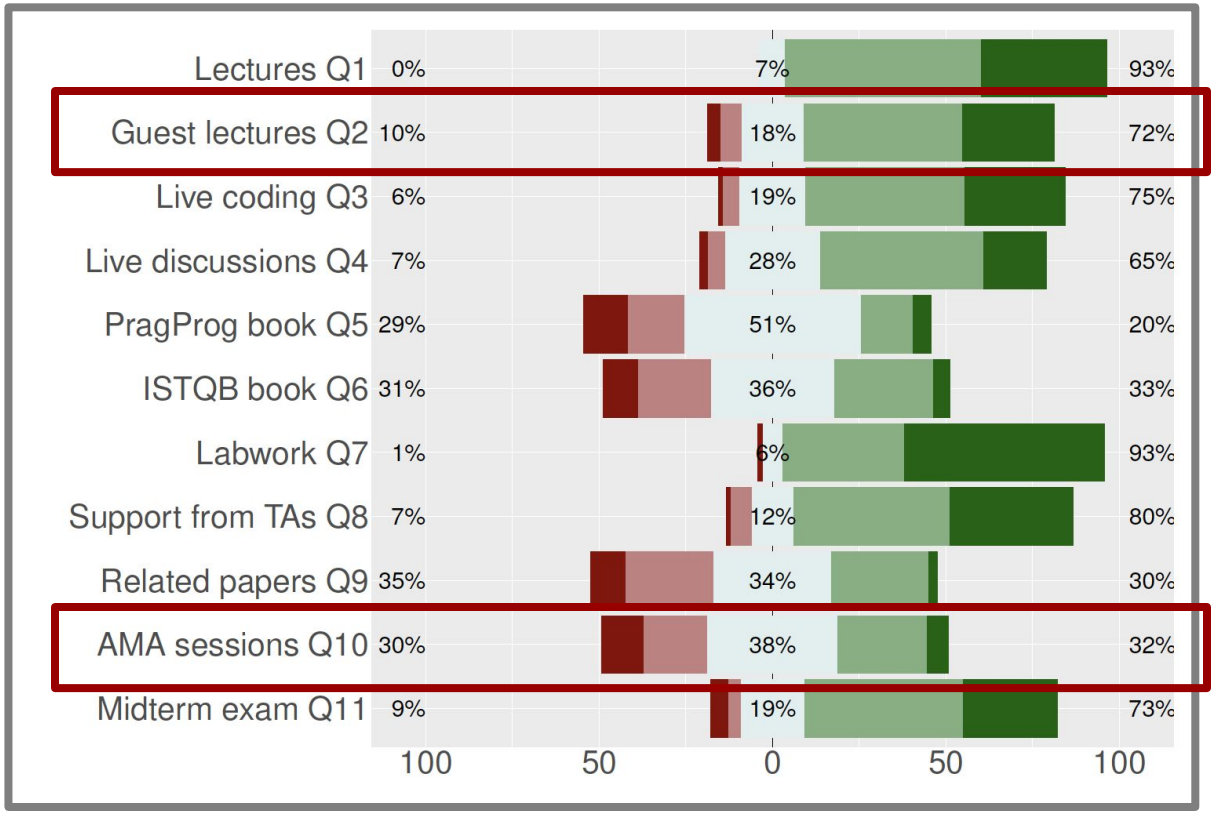
However, when it comes to best practice, although students overall perceive it as easy, TAs disagree. This also contradicts data in RQ1.

Favourite learning methods



We still lack books that students can enjoy...

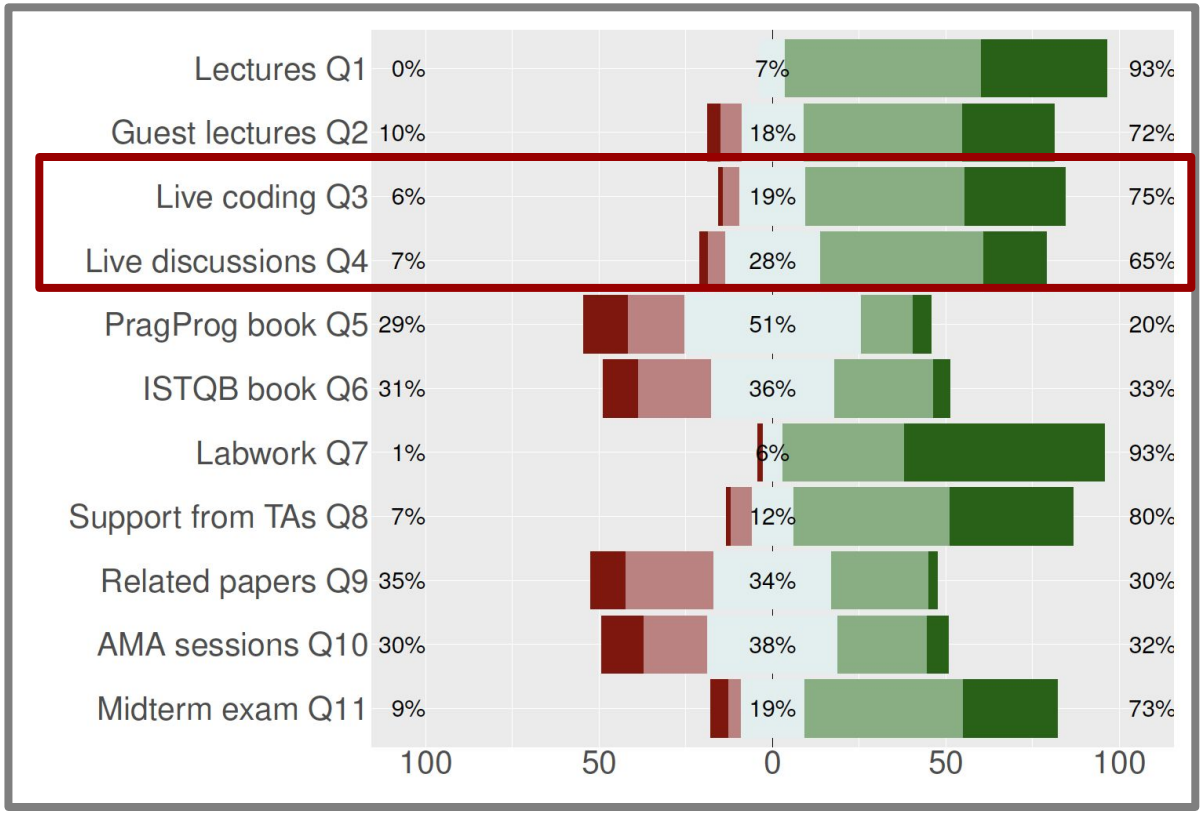
Favourite learning methods



They enjoy guest lectures. However, they did not enjoy AMA as much as we'd have hoped.

We conjecture it's due to their lack of experience, i.e., it's hard for them to come up with questions for practitioners.

Favourite learning methods



Live coding and discussions are appreciated.

Where we are now?

- Better balance between formative and summative assessments.
- Scalability issues in assessment:
  - Try self-assessment.
  - MC questions are not my favorite option.
- Increase the testing challenges in JPacman
- Write lecture notes (open book)

Following

Nat Pryce
@natpryce

Pragmatic Software Testing Education:
'Questions like “Do I need to test this behavior via an unit or a system test?”, “How can I test my mobile application?” are … discussed not only through the eyes of software testing, but also … software design.' 👏
pure.tudelft.nl/portal/files/4 …

9:09 AM - 25 Feb 2019

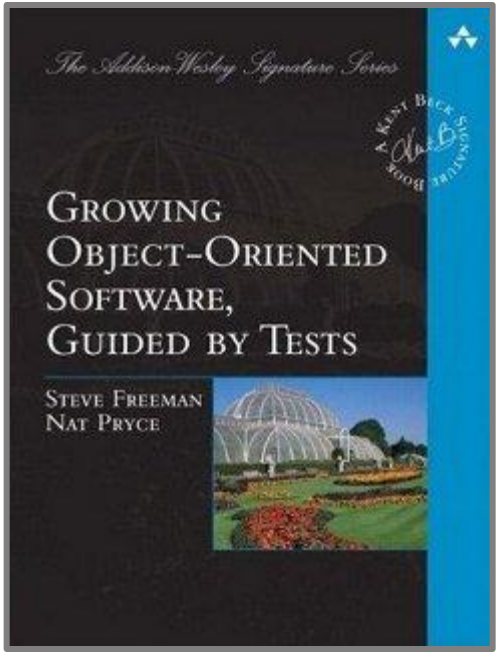
13 Retweets **61** Likes

5 13 61

Tweet your reply

Nat Pryce @natpryce · Feb 25
Paper by Maurício Aniche, Felienne Hermans & Arie van Deursen of Delft University of Technology.

4



Pragmatic Software Testing Education

Maurício Aniche - m.f.aniche@tudelft.nl

Felienne Hermans - f.f.j.hermans@liacs.leidenuniv.nl

Arie van Deursen - arie.vandeursen@tudelft.nl

TU Delft

Photos in this presentation

- Photo by Houcine Ncib: https://unsplash.com/photos/jEYwJ4MYcAM
- Photo by Michał Parzuchowski: https://unsplash.com/photos/geNNFqfvw48
- Photo by 贝莉儿 NG: https://unsplash.com/photos/kWvsIYwUkJs
- Photo by Liam Welch: https://unsplash.com/photos/mVS69QbE4H4
- Photo by adrian: https://unsplash.com/photos/yocwPcXJe9c
- Photo by Dane Deaner: https://unsplash.com/photos/JNpmCYZID68
- Photo by John Doyle: https://unsplash.com/photos/RSgwLqIWH8w
- Photo by Markus Spiske: https://unsplash.com/photos/tGcCXy5bfzQ
- Photo by Elisa Michelet: https://unsplash.com/photos/b4EsL48DIK0
- Photo by Jon Tyson: https://unsplash.com/photos/TBtsrLYebSU
- Photo by Damir Bosnjak: https://unsplash.com/photos/W5qJExlQTGI