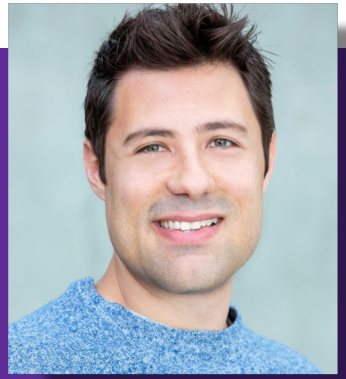
Pragmatic Software Testing Education

(experience report)

Maurício Aniche, Felienne Hermans, Arie van Deursen

TU Delft



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

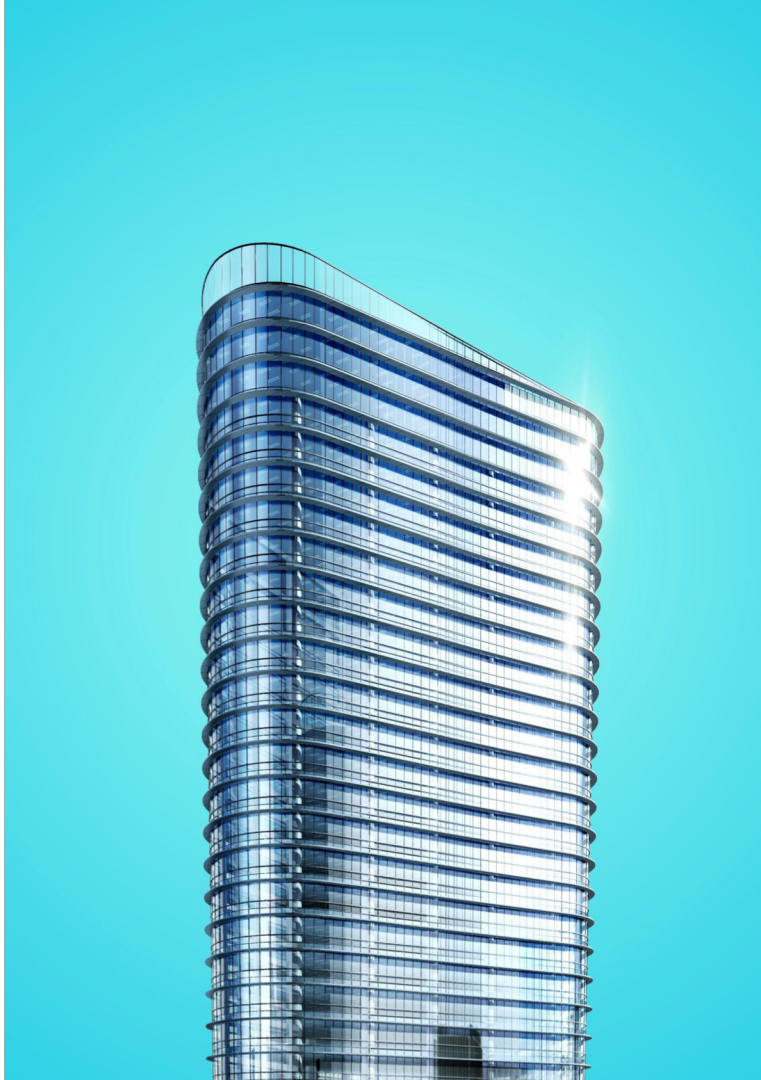
- xUnit frameworks
- The Testing Pyramid
- Mocking
- What to test? What not to test?

How to combine both perspectives?

(This list was not developed in a systematic way)

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.



If we look at our data...

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

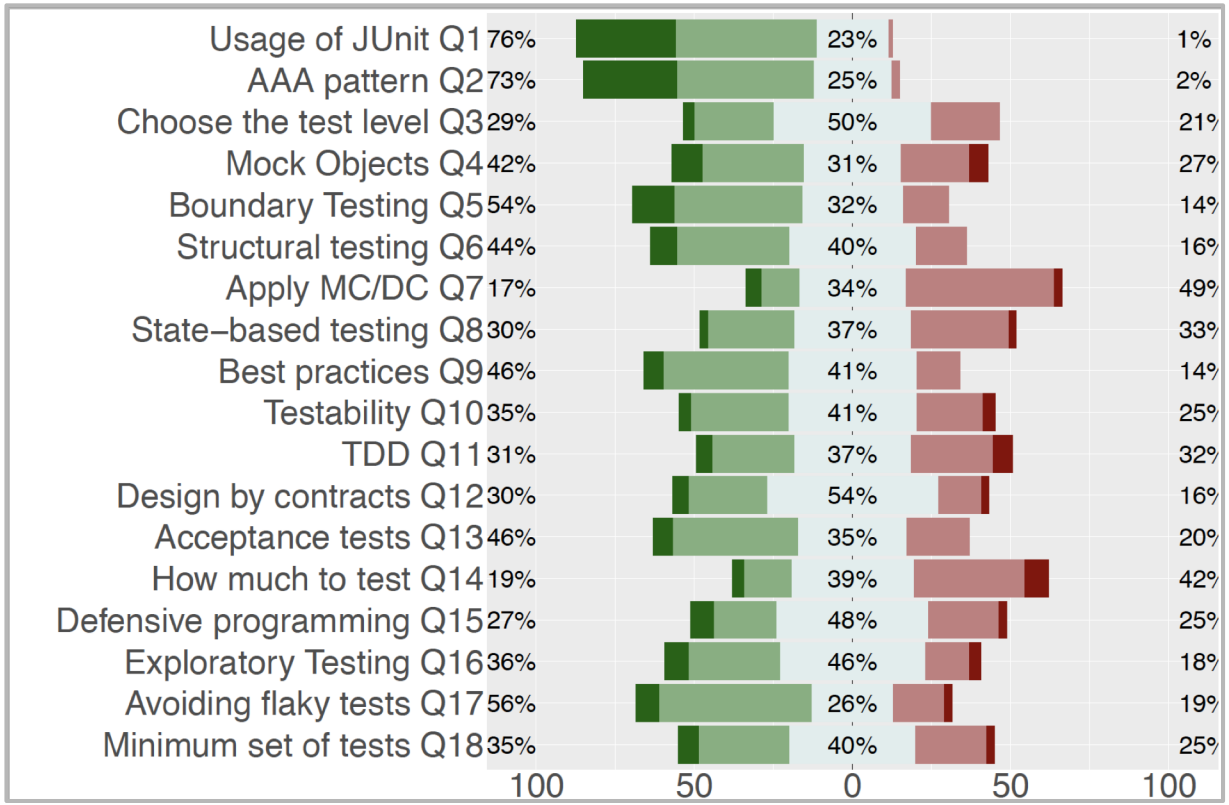
Their common mistakes

- 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.

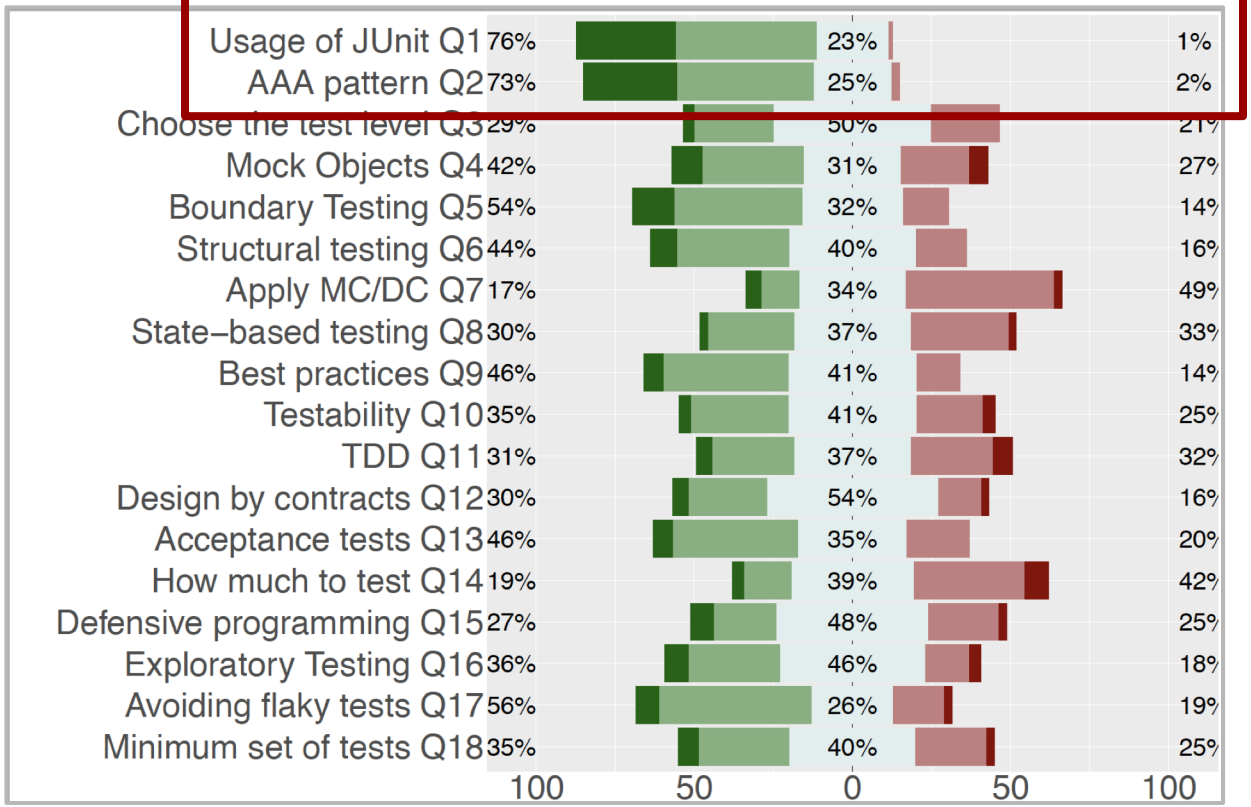
Their common mistakes

- 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



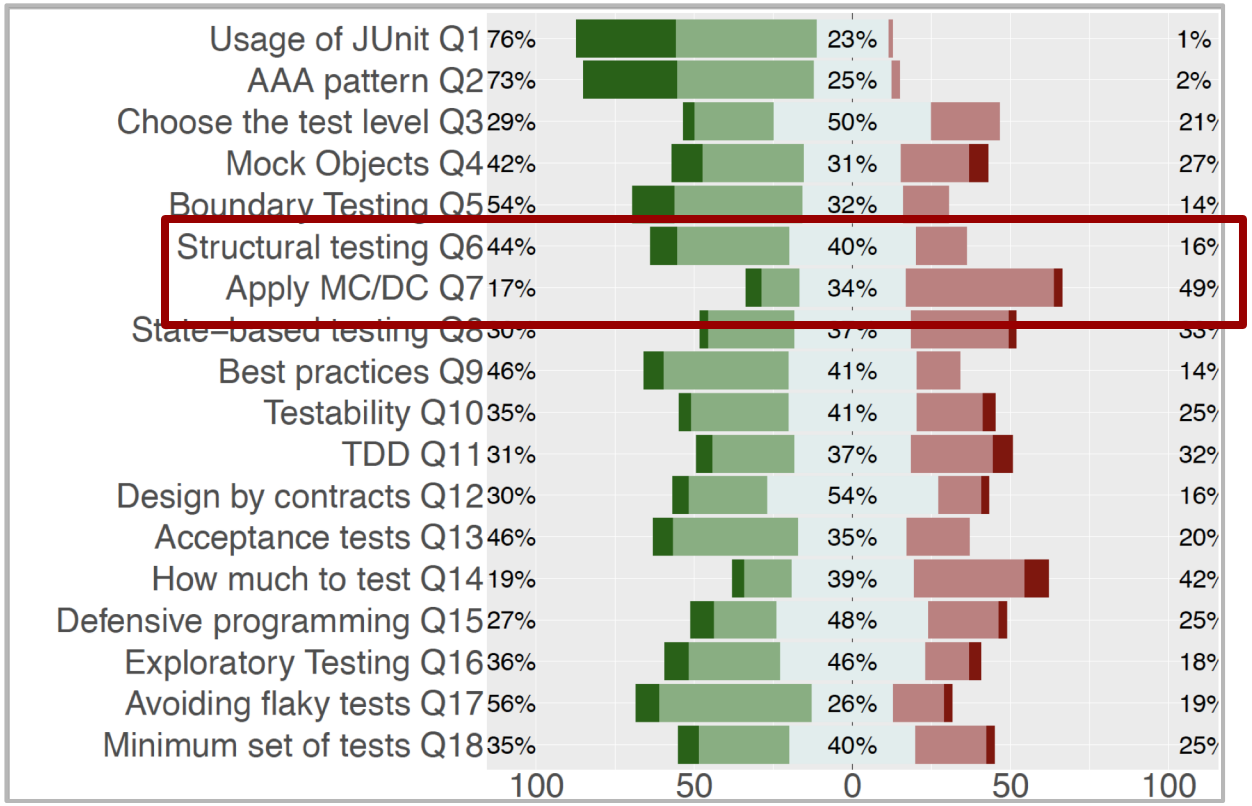
Topics hard to learn



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.

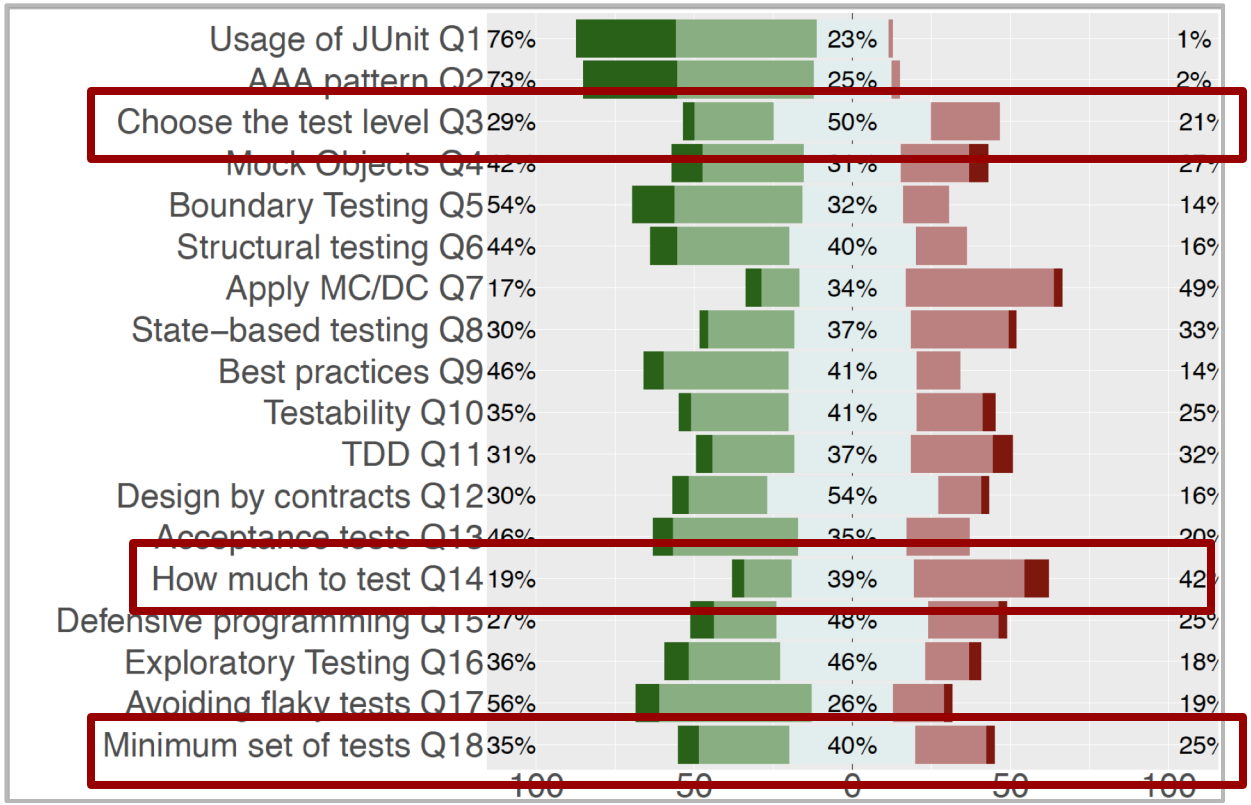
(Matches the number of feedback related to tools in previous RQ)

Topics hard to learn



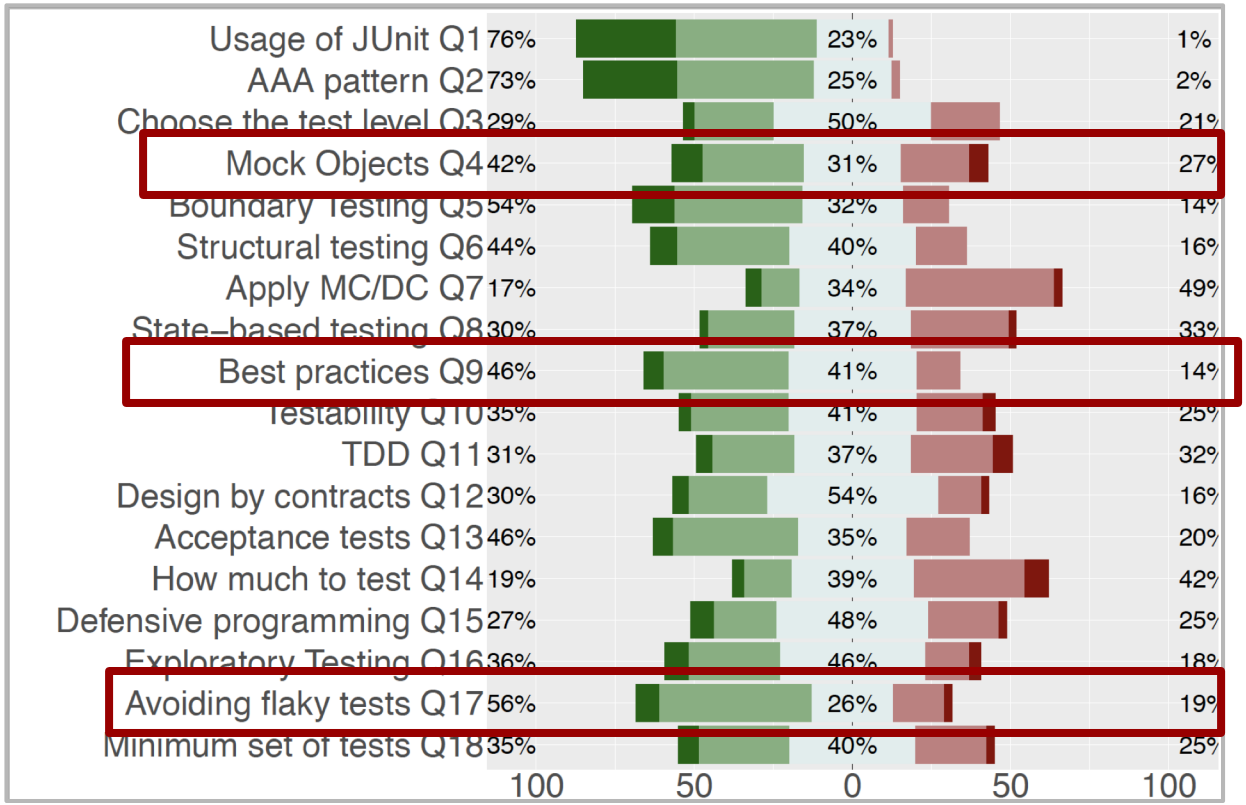
MC/DC is not an easy coverage criteria. However, structural testing in general was considered a somewhat easy topic.

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.

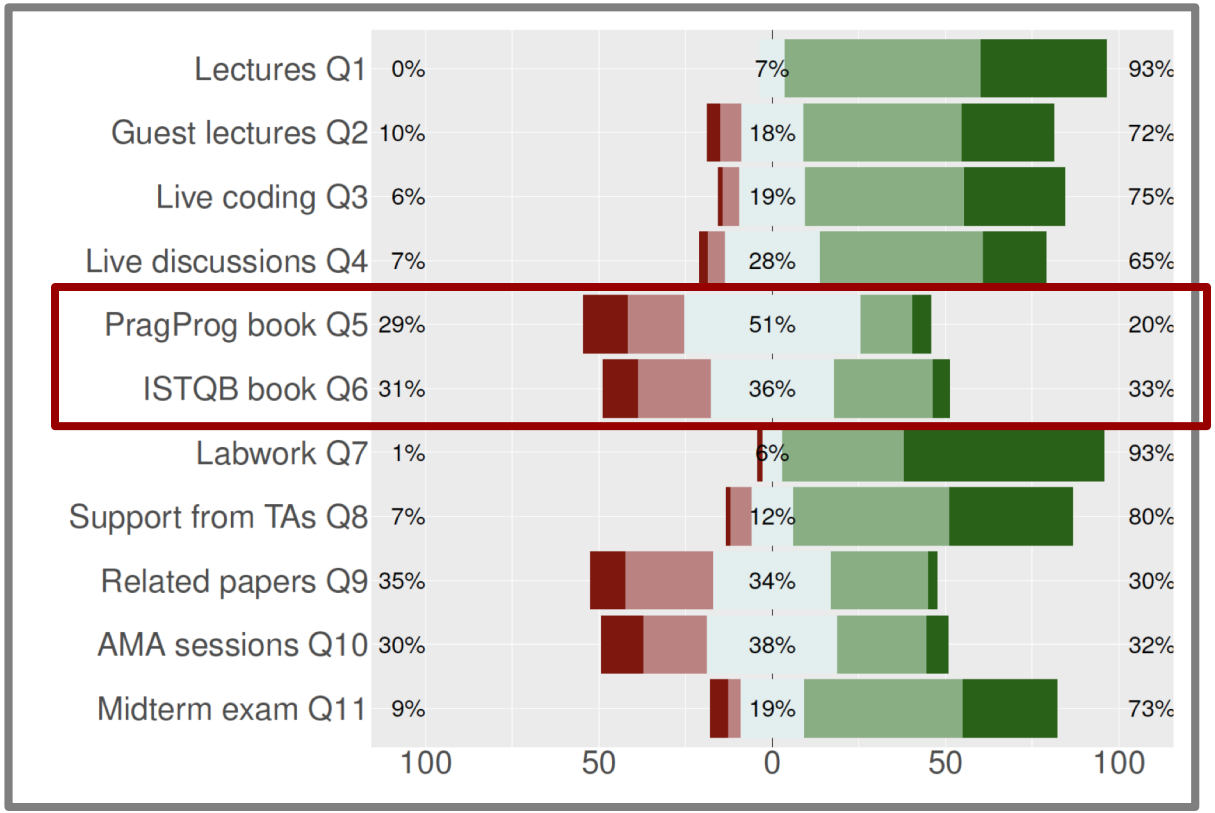
Topics hard to learn



Students think Mock Objects are an easy topic.

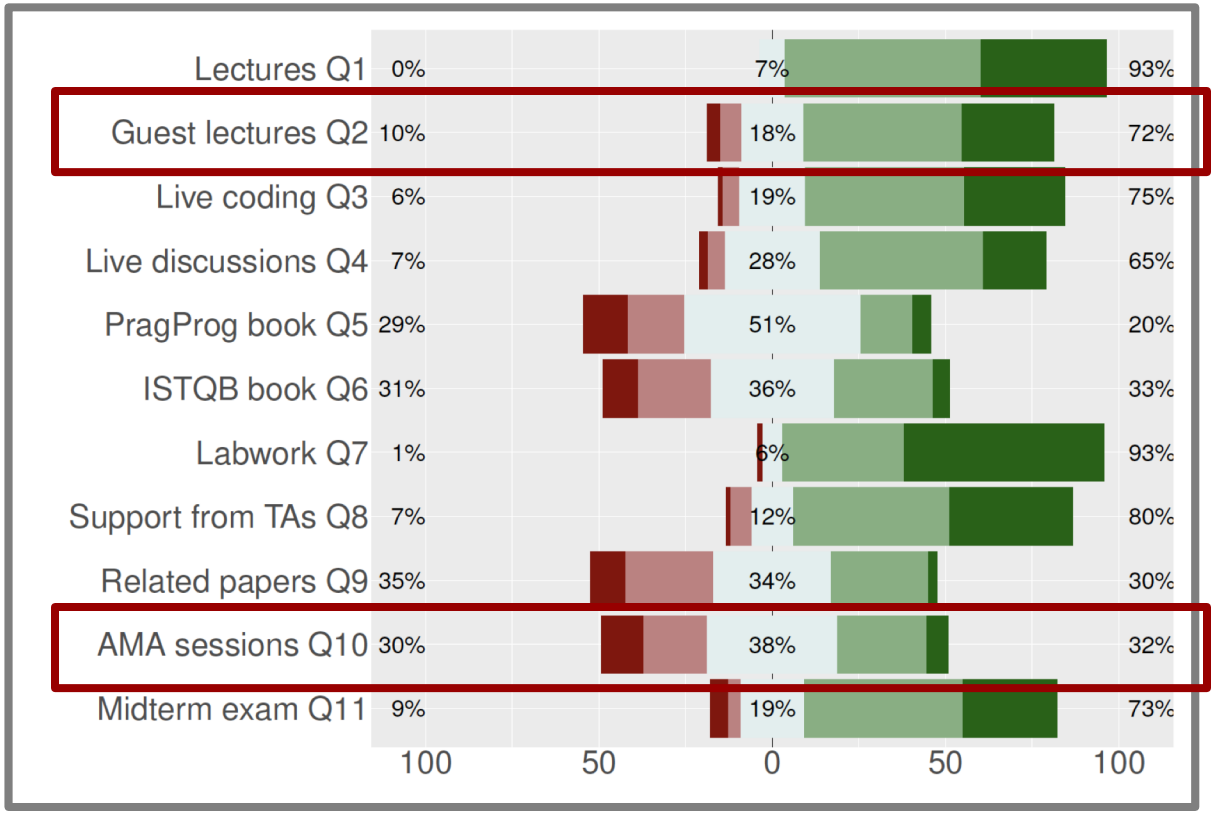
However, when it comes to best practice, although students overall perceive it as easy, TAs disagree. This also contradicts data in RQ1.

Favourite learning methods



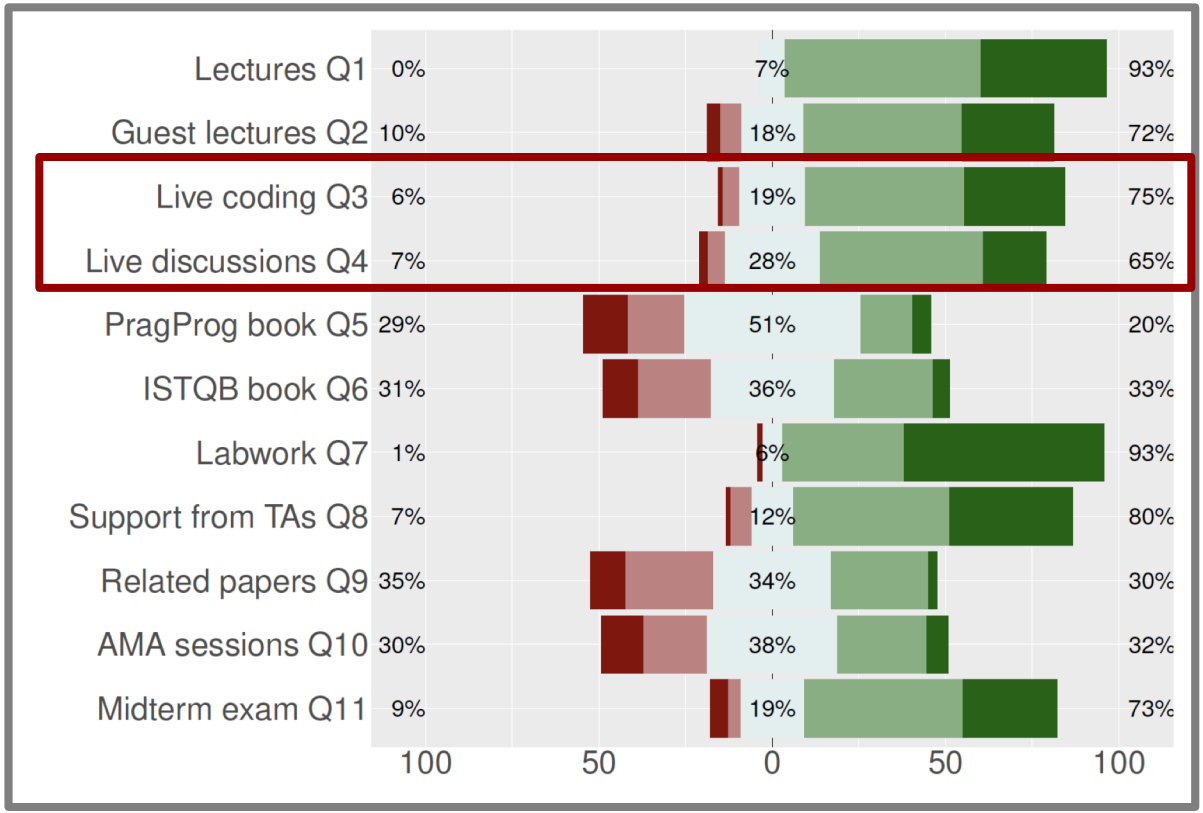
We still lack books that students can enjoy...

Favourite learning methods



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

Favourite learning methods



Live coding and discussions are appreciated.

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