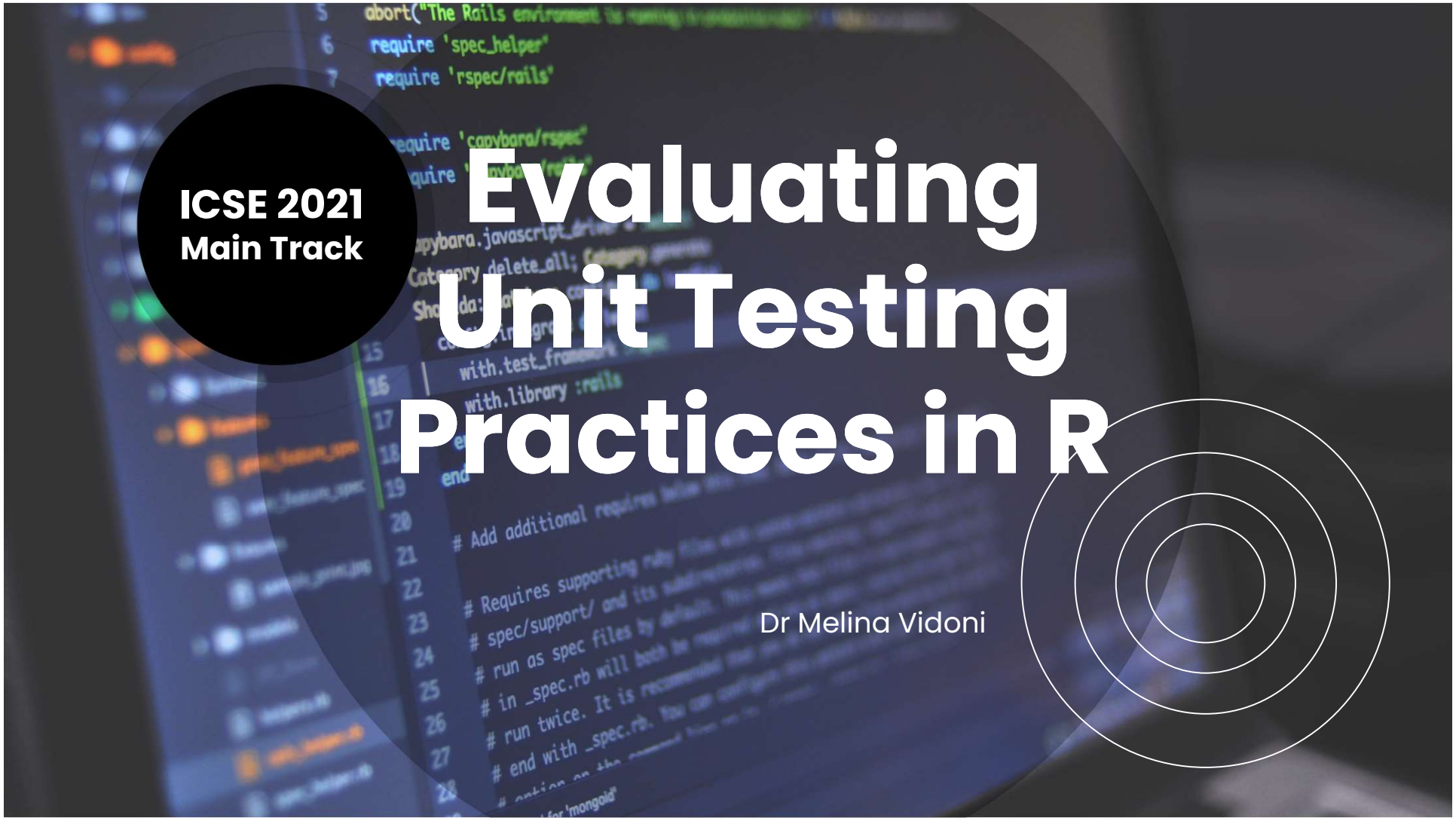
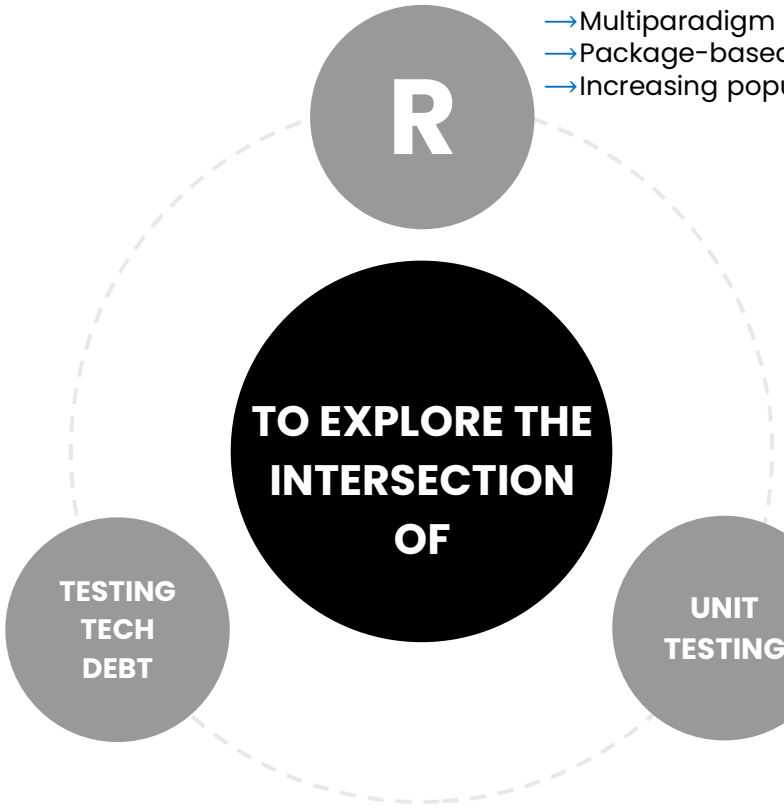

ICSE 2021
Main Track
Evaluating Unit Testing Practices in R
Dr Melina Vidoni

Motivation

Testing Technical Debt (TTD) occurs due to shortcuts (non-optimal decisions) taken about testing.



→Open-source
→Statistical and data science focus
→Multiparadigm
→Package-based
→Increasing popularity

→No previous research about testing in R packages.

→Need to understand how testing is approached.

→Need to understand developers' challenges to provide solutions.

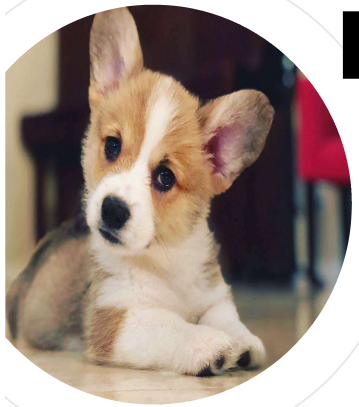
RMIT

RELATED WORKS

- → GitHub influences in the R ecosystem, re: distribution of packages and inter-repository dependencies (Decan et al., 2016)
- → Studying R package maintenance to explore frequency of change (Ramirez et al., 2015)
- → Evaluation of R features to understand language design (Morandat et al., 2012).

- → MSR+Survey to assess the popularity of open-source GitHub Repositories, using stars and watches (Borges et al., 2018)
- → MSR+Survey to evaluate why GitHub respositories are forked (Jiang et al., 2017)
- → MSR+Survey to determine TTD in Scala projects, identifying testing smells (Bleser et al., 2019).

- → Inspecting the code of R packages to create a tool that automatically generates unit tests; empirically evaluated only (Krikava et al., 2018).



RESEARCH QUESTIONS

RQ1. Are R packages well tested? To understand which testing tools are used in R packages, identify common practices, types of testing, and how unit testing tailors to a multi-paradigm language like R.

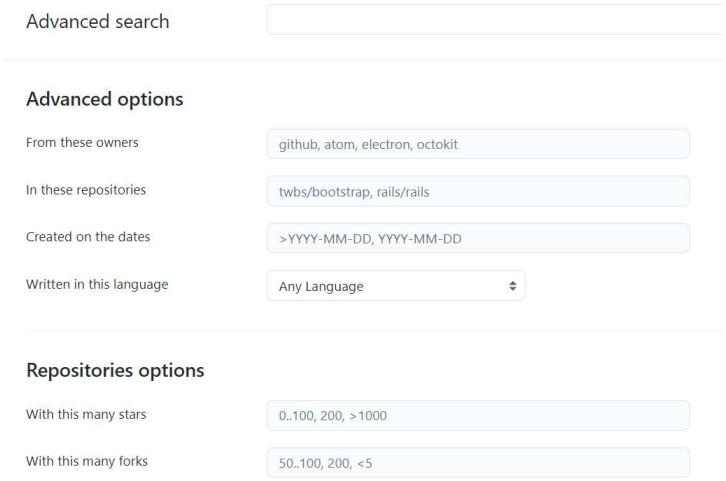
RQ2. Which are potential Testing TD weak-spots? To discover and understand negative practices that affect unit testing in R packages. The long-term goal of this is to identify testing smells.

RQ3. How do R package developers perceive unit testing? Part of the MSR involved collecting public email addresses of developers, disclosed in packages' files, to send them a structured survey. Questions aimed to understand their subjective perception of testing and the challenges they face.

MSR (PART 1)

Inclusion Criteria: The repository must be an R-package, originally posted during or after 2010; it needs to show maintenance activity (commits) in the last two years (i.e. from 2018). It must have a correct package structure, with all dependencies available.

Exclusion Criteria: The repository is an R data package, a book, or a personal package. The state of the repository is archived, deprecated, or outdated. It is an R package with scripts used in a book. It has incomplete or missing files (i.e. description, namespace, or readme files). It is a fork from another R package.



RMIT

MSR (PART 1)

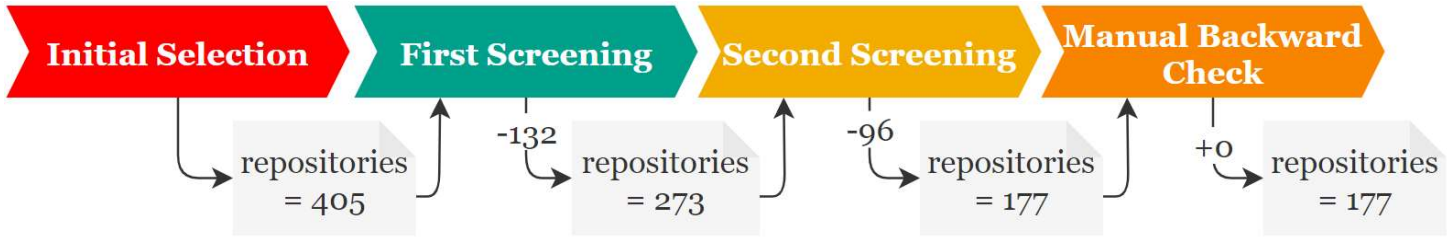


TABLE I
SECOND SCREENING FILTERING RESULTS, WITH `covr` AND `testthat` RESULTS.

`covr`	`testthat`	Result	Number
Yes	Yes	Both analysis run correctly	159
Yes	Failed	`covr` runs, but there are issues with `testthat`	18
NA	NA	Analysis are unable to run. Empty test structure, or manual test cases	45
NA	Manual		20
Error	Yes	Filtered. `covr` is unable to complete the analysis. `testthat` provides mixed results	19
Error	Failed		6
Error	NA		6

DEVS SURVEY (PART 2)

- Implemented in Qualtrics
- Email information removed to ensure anonymity of respondents
- Emails obtained from R packages “Description” file, field Authors@R
- 469 email addresses, 22 emails bounded, 91 replies collected (19.4% response rate)
- Ethical Approval required not publishing the data set

TABLE II
STRUCTURE OF THE SURVEY GENERATED FOR PART II.

Question	Possible Answers
How many R packages have you authored? (Regardless if they are in CRAN/Bioconductor or not)	<2 packages / 2-5 / 5-10 / >10
How many years of experience do you have as an R programmer?	<2 years / 2-5 / 5-10 / 10+ years
How do you test your code? [S]	Manually / I don’t test / Using testing packages
What type of testing do you do?	Individual Functions Only / Functions Clusters / Using my package externally / Other
If you use testing packages, what are the names of them?	Comment box
Why do you use testing packages?	Generating or executing test cases / Creating and evaluating results / Analysing code coverage / Finding bugs / Reporting bugs / Fulfilling CRAN requirements.
Do you face the following challenges during testing? And if you do, how serious are they?	Likert Scale 1-5
What are the top two things you look for/need/would like to see?	Comment box
Do you use coverage visualisation tools? [S]	Always / Occasionally / Never
Name the coverage visualisation tools that you use.	Comment box
Does coverage visualisation affect you? [S]	It motivates me / It makes me anxious / It makes me confident in my code / I trust my code is bug-free / Other
Did you ever have all tests passing, but found a bug in your code? [S]	Yes, at least once / Yes, more than one time / I don’t remember / Never

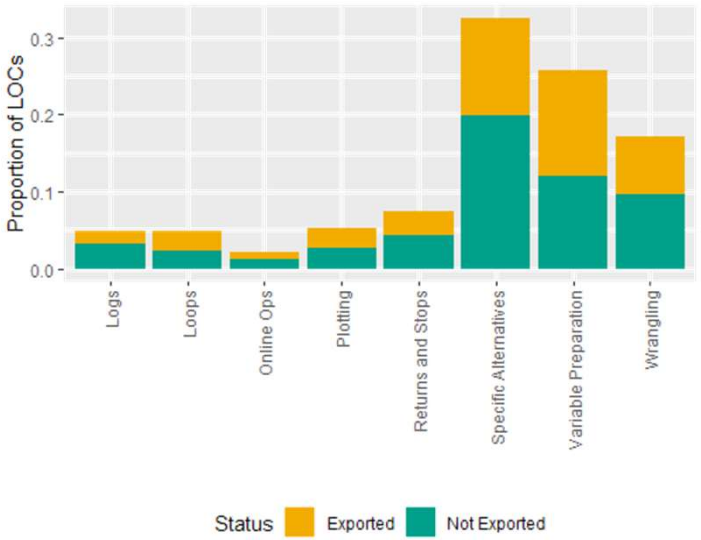
Testing Quality (RQ1, RQ2)

TESTING COVERAGE & RELEVANT LINES

- Classifying the package by discipline and type, according to what they stated in the Readme.md
- Automated *covr* analysis to compared tested lines
- ***Covr* determined: 40% are relevant lines (only 43% are tested!)**
- Average coverage: 48.6%

URLOC = UNTESTED RELEVANT LINES OF CODE

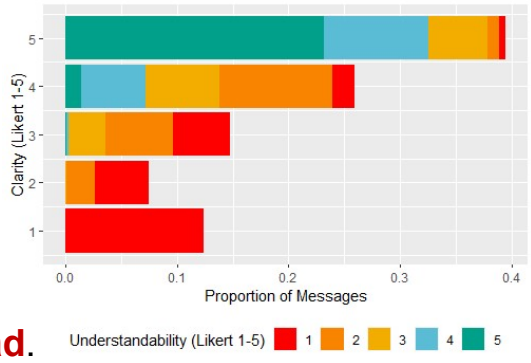
- A sub-sample was manually classified by goal
- 95% confidence, 3% error = 1416 lines
- 55.3% of URLOCs belong to non-exported functions.
- The most representative group *are specific alternatives* (32.6%). Smelly! Not all paths are appropriately tested.



SUMMARY OF ANALYSIS

INFORMATIVE ASSERTS

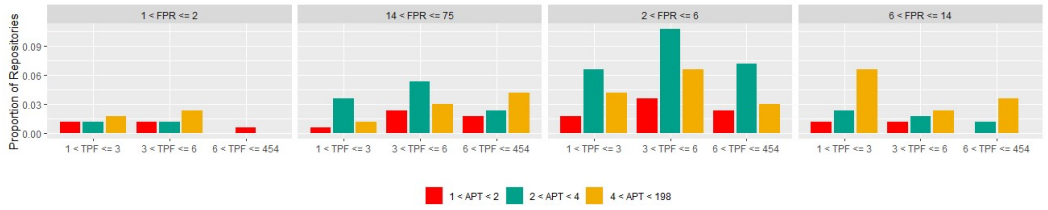
- R-script to automatically pre-classify asserts.
- 98% have a written message.
- Average message length: 3~7 words.



- Manual classification in sub-sample of 1416 messages.
- Clarity: language semantics. 40% = very clear, <20% = unclear.
- **Understandability: what is being tested. ~45% are challenging to read**.

ORGANISATION OF TEST FILES

- R script to crawl source code, extracting signatures of test methods, assertions and LOC position.
- Done per package, per folder, per file.
- Most repost have between 2~5.5 test files, regardless of the size of the code.
- In the above group: 3~6 test methods per test file.
- Also: 2~4 assertions in each test method.



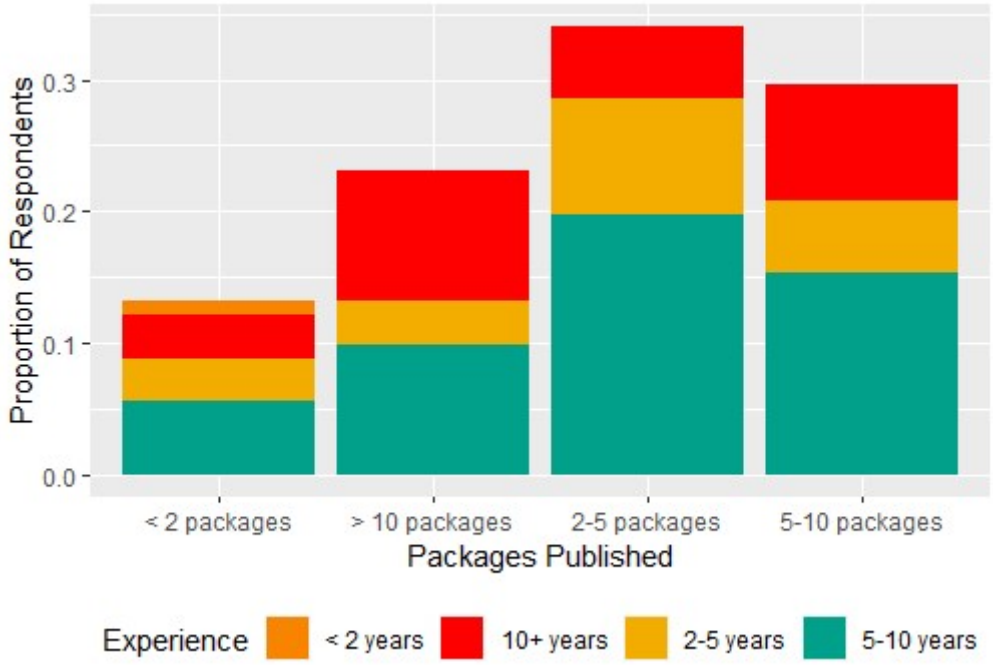
SUMMARY OF ANALYSIS

TYPES OF ASSERTS

- Only 1% of developers used something that was not *testthat* (from the survey)
- *Testthat* has no @beforeall or @beforeeach equivalent. Many tests fail during variable initialisation.
- *37166* unique assertions detected => 80.2% are custom defined

- Manual study in subsample to classify in common/edge/dummy
- Only 3% are manual tests, and all of them evaluated plots
- About 82.5% of asserts evaluate common cases => few edge cases being tested

Devs Survey (RQ3)



About half respondents have between 5-10 years of experience as R developers,

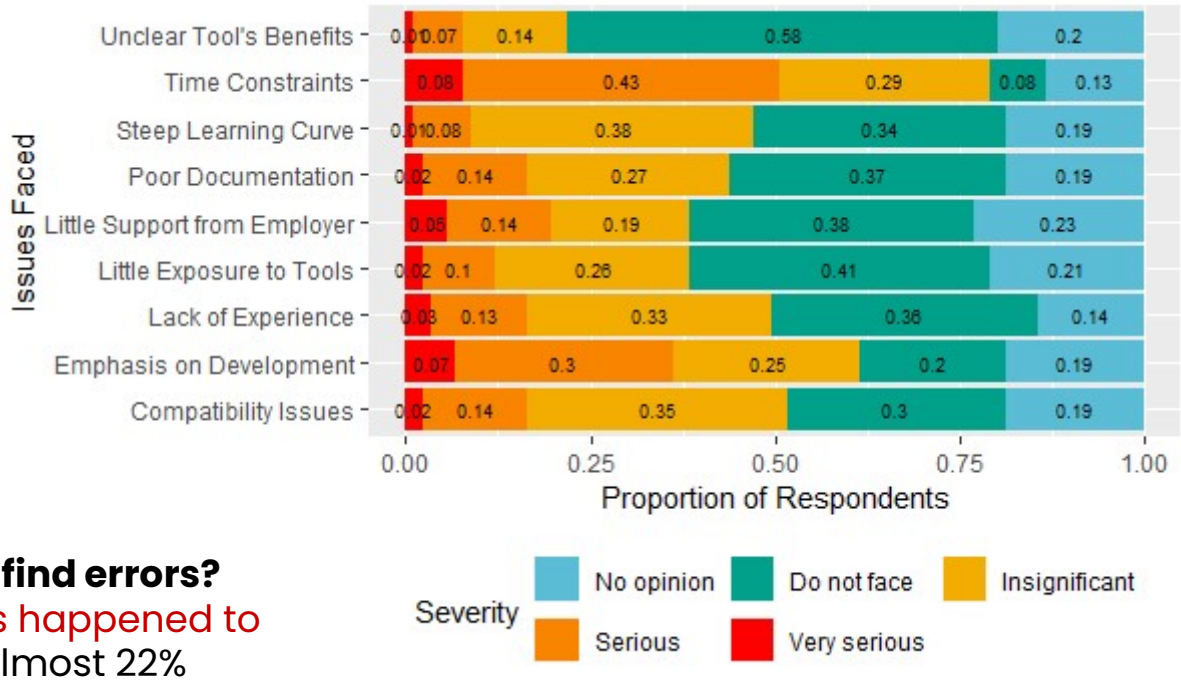
Almost 27.5% have 10+ years of R programming experience.

About 29.7% had between 5-10 R packages published,

About 23% had more than ten packages.

SELF REPORTED DEMOGRAPHICS

Almost 16% declared *lack of testing experience* as a Serious/Very Serious issue, with similar severity regarding to *poor documentation*. Furthermore, almost 10% still face *steep learning curve* for unit testing.



All tests are passing and you still find errors?
Close to 58% participants said this happened to them “more than one time", and almost 22% estimated “at least once"; almost 9% “did not remember".

TESTING CHALLENGES

SUMMARY OF RESULTS

TABLE VII
TYPES OF TTD, SMELLS, AND RESULTS SHOWCASING WEAK-SPOTS

Type	Smell	Reason
Unit Testing	Inadequate Unit Tests	Elevated number of relevant lines still untested (see Table III and Figure 3).
		Many alternative paths, belonging to exported functions, are not being tested (see Figure 3).
		Elevated variability of coverage between packages of the same discipline. This may indicate incomplete or excess testing (see Figure 2).
		Increased focus on testing common cases, with little focus on assessing edge cases (see Figure 7).
	Obscure Unit Tests	Though many asserts have messages, they are mostly unclear and not understandable (see Figure 4).
		In average, there are too many asserts per test method, lowering the readability of automated testing results (see Table IV and Figure 5).
		Excessive use of custom asserts may hinder testing understandability (see Section III-A6).
	Improper Asserts	Too many common cases are being tested, and few common cases are being evaluated (see Figure 7).
		Excessive use of custom asserts may indicate potential issues with testing frameworks and developers training (see Section III-A6).
		Developers finding bugs regardless of having test suites with all test passing (see Section III-B2).
Exploratory Testing	Inexperienced Testers	Though most survey participants reported a high level of expertise (see Figure 8), their main concern in terms of improvement for existing tools was better documentation, tutorials and examples, as well as guides to create meaningful tests for data science. This is also supported by the indicated severity (medium-to-high) of challenges such as steep learning curve, and poor documentation.
Manual Testing	Limited Test Execution	About 20 papers were filtered as they included only manual testing cases, with no unit testing.
		About 12% of survey participants acknowledged performing only manual testing in their packages (see Section III-B1).
	Improper Test Design	About 3% of asserts were determined to be manual, as they were always testing plots. Though the number is small, there was also a low amount of plotting-related R packages in the selected sample. As plotting and visualisation are vital for data science [25], better testing tools should be developed.

RQ1/RQ2: IDENTIFIED SMELLS

RMIT

Lack of training in developers. Besides self-reported issues on the survey carried out in this study, previous research also demonstrated that most R programmers come from diverse technical backgrounds not focused on programming [10]

Incomplete tools due to the towering number of custom asserts, challenges such as compatibility issues, and desired improvements such as better automation, test data generation, and comparison between testing suits. This is also supported by the lack of methods that could be used to initialise test data.

RQ3: DEVS' CHALLENGES

RMIT

R package testing cannot be considered comprehensive or high-quality. Several reasons support this: many alternative paths are not being tested, there is a highly variable coverage, and the occurrence of manual testing.

Several TTD smells have been identified by comparing the results of the study to existing TTD smells classifications. *Common smells are: inadequate and obscure unit tests, improper asserts, inexperienced testers, and improper test design.*

R packages developers face numerous challenges. Participants of the survey self-reported a high level of expertise. However, they agreed on the following challenges: time constraints, emphasis on development rather than testing, poor documentation of tools, steep learning curve, and still finding bugs despise of having test suits with all-passing tests.

CONCLUSIONS

RMIT

1) Analysing other types of debt, like SATD.

2) What is the ideal coverage for R packages?

3) What is the impact of testing (or not testing) non-exported functions?

4) How to improve existing unit testing tools?

FUTURE WORKS

RMIT

Thanks!