



SOFTWARE TOOL ARTICLE

REVISED *exampletestr*—An easy start to unit testing R packages

[version 2; referees: 2 approved]

Previously titled: Easier unit tests and better examples with *exampletestr* and *covr*

Rory Nolan¹, Sergi Padilla-Parra^{1,2}

¹Wellcome Trust Centre for Human Genetics, University of Oxford, Oxford, OX3 7BN, UK

²Department of Structural Biology, University of Oxford, Oxford, OX3 7BN, UK

v2 First published: 17 May 2017, 2:31 (doi: [10.12688/wellcomeopenres.11635.1](https://doi.org/10.12688/wellcomeopenres.11635.1))
 Latest published: 21 Jun 2017, 2:31 (doi: [10.12688/wellcomeopenres.11635.2](https://doi.org/10.12688/wellcomeopenres.11635.2))

Abstract

In spite of the utility of unit tests, most R package developers do not write them. *exampletestr* makes it easier to *start* writing unit tests by creating shells/skeletons of unit tests based on the examples in the user's package documentation. When completed, these unit tests test whether said examples run *correctly*. By combining the functionality of *exampletestr* with that of *covr*, having ensured that their examples adequately demonstrate the features of their package, the developer can have much of the work of constructing a comprehensive set of unit tests done for them.

Open Peer Review

Referee Status:

	Invited Referees	
	1	2
version 2 published 21 Jun 2017		 report
version 1 published 17 May 2017	 report	 report

- 1 **Thomas Quinn** , Deakin University, Australia
- 2 **Laurent Gatto** , University of Cambridge, UK

Discuss this article

Comments (0)

Corresponding author: Rory Nolan (rnolan@well.ox.ac.uk)

Author roles: **Nolan R:** Conceptualization, Data Curation, Formal Analysis, Funding Acquisition, Investigation, Methodology, Resources, Software, Validation, Writing – Original Draft Preparation, Writing – Review & Editing; **Padilla-Parra S:** Funding Acquisition, Project Administration, Resources, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

How to cite this article: Nolan R and Padilla-Parra S. *exampletestr—An easy start to unit testing R packages [version 2; referees: 2 approved]* Wellcome Open Research 2017, 2:31 (doi: [10.12688/wellcomeopenres.11635.2](https://doi.org/10.12688/wellcomeopenres.11635.2))

Copyright: © 2017 Nolan R and Padilla-Parra S. This is an open access article distributed under the terms of the [Creative Commons Attribution Licence](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Grant information: This work was been supported by the Wellcome Trust [105278] and [203141].

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

First published: 17 May 2017, 2:31 (doi: [10.12688/wellcomeopenres.11635.1](https://doi.org/10.12688/wellcomeopenres.11635.1))

REVISED Amendments from Version 1

This version (2) makes revisions suggested by reviewer Laurent Gatto. The introduction now provides the rationale for unit testing. A discussion is included to stress the point that the type of tests that `exampletestr` can help you with are not the only type of test that a package should have. It is made clear that retrospective testing (which is what `exampletestr` is useful for) is not necessarily the best way to test a package.

See referee reports

Introduction

Unit tests are automated checks which examine whether a package works as *intended*. There are many reasons to write unit tests:

1. **To identify bugs in your current code.** If a test fails, something is not working as expected.
2. **To make current package functionality robust to future changes.** If you have written good tests, you do not need to worry that changes made to your code have broken existing functionality; the tests will tell you whether or not the original functionality is in tact. In the words of Hadley Wickham, "I started using automated tests because I discovered I was spending too much time refixing bugs that I'd already fixed before."¹
3. **Test-driven development (TDD).** Some people advocate writing tests before writing code: the tests outline what your code should do and you know your code is working correctly when the tests start passing. The use of TDD achieves the quality control goals of points 1 and 2 above as code is written. This contrasts with *retrospective unit testing*, which is to write unit tests after everything else is done.
4. **To assure others that your code works correctly.** The fact that a package is unit tested indicates to potential users that the author has taken care to ensure that their code works as intended.

For the above reasons, it is good practice to write unit tests for R packages. However, at the time of writing, of the 10084 packages on CRAN, only 25% (2566) are unit tested. Of the 5715 packages authored or updated since 1st January 2016, 35% (1984) are unit tested. Hence, the majority of packages on CRAN are not unit tested. Testing for these packages would have to be done retrospectively.

This paper describes `exampletestr`, a package that makes it easy for package developers to retrospectively create unit tests based on the examples in their package documentation. `exampletestr` is designed for developers who have a functioning package

that they would like to unit test; it is less useful for those using TDD.

Methods

Implementation

`exampletestr` is designed for tests to be written within the `testthat`² framework. This is the most popular testing framework, preferred in 93% of cases. `exampletestr` works by reading the examples in the `.Rd` files in the `man/` folder of the package project. These examples are then wrapped in `test_that` and `expect_equal` statements, thereby creating *test shells*.

By running `make_tests_shells_pkg()` in the root directory of a package, for each `x.R` file in the `R/` directory that has at least one function documented with an example, you get a corresponding file `test_x.R` in the `tests/testthat/` directory of the package, containing these test shells to be filled in by the user (a package developer) to create fully functional unit tests.

For a complete overview of `exampletestr`'s capabilities, consult the package's manual and vignette at <https://CRAN.R-project.org/package=exampletestr>.

Operation

This package can be used with R version 3.3.0 or later on Linux, Mac and Windows. It can be installed from within R via the command `install.packages("exampletestr")`.

The idea of basing unit tests around documentation examples suggests the use of `covr`³ in the following way to ensure both that the examples in the documentation exhibit as much of the functionality of the package as possible and that the unit tests cover as much of the code as feasible:

1. Write comprehensive documentation examples for your package, using `covr`'s `package_coverage(type = "examples") %>% shine()` to ensure that all sections of code that the package user may find useful are demonstrated.
2. Write unit tests corresponding to *all* of your examples using `exampletestr`.
3. Complete the writing of unit tests, checking your code coverage with `package_coverage(type = "tests") %>% shine()`.

Using this workflow, the developer ensures that their *example coverage* (the portion of package features covered by documentation examples) is adequate, and simultaneously obtains a reduction in the work required to write comprehensive unit tests.

Use cases

The best way to display the functionality of `exampletestr` is by example. Take the `str_detect` function from the `stringr`

package⁴. The main file `str_detect.Rd` has the examples section:

```
\examples{
fruit <- c("apple", "banana", "pear", "pinapple")
str_detect(fruit, "a")
str_detect(fruit, "^a")
str_detect(fruit, "a$")
str_detect(fruit, "b")
str_detect(fruit, "[aeiou]")
# Also vectorised over pattern
str_detect("aecfg", letters)
}
```

The *test shell* that would be automatically created by `exampletestr` from these examples is:

```
test_that("str_detect works", {
  fruit <- c("apple", "banana", "pear", "pinapple")
  expect_equal(str_detect(fruit, "a"), )
  expect_equal(str_detect(fruit, "^a"), )
  expect_equal(str_detect(fruit, "a$"), )
  expect_equal(str_detect(fruit, "b"), )
  expect_equal(str_detect(fruit, "[aeiou]"), )
  expect_equal(str_detect("aecfg", letters), )
})
```

which can then be filled in by the package developer to give the complete and passing test:

```
test_that("str_detect works", {
  fruit <- c("apple", "banana", "pear", "pinapple")
  expect_equal(str_detect(fruit, "a"), rep(TRUE, 4))
  expect_equal(str_detect(fruit, "^a"),
               c(TRUE, rep(FALSE, 3)))
  expect_equal(str_detect(fruit, "a$"),
               c(FALSE, TRUE, FALSE, FALSE))
  expect_equal(str_detect(fruit, "b"),
               c(FALSE, TRUE, FALSE, FALSE))
  expect_equal(str_detect(fruit, "[aeiou]"),
               rep(TRUE, 4))
  expect_equal(str_detect("aecfg", letters),
               c(TRUE, FALSE, TRUE, FALSE, TRUE,
                 TRUE, TRUE, rep(FALSE, 19)))
})
```

Discussion

Testing whether documentation examples run correctly (which is what `exampletestr` is useful for) is just one important part of writing good unit tests. Two points to bear in mind when using `exampletestr`:

1. Documentation examples typically showcase a *best case* application of the code in a package. Unit testing should

go beyond this and assess the execution of edge cases, e.g. the performance of functions when an argument is empty (length zero). `exampletestr` does not help in this regard.

2. `exampletestr` promotes a “one test per function” model. However, in many packages, it is necessary to go further than just testing whether each function performs well in isolation. For instance, it is good to test function composition: whether one obtains the expected results when applying two or more functions, one after another.

Conclusions

Unit tests are crucial to ensuring that a package functions correctly, yet most developers do not write them. Most package developers do, however, write documentation examples. With `exampletestr`, documentation examples are easily transformed into unit tests; thereby encouraging maintainers of untested packages to make a start on unit testing.

Software and data availability

`exampletestr` can be downloaded from: <https://CRAN.R-project.org/package=exampletestr>

Source code available from: <https://github.com/rorynolan/exampletestr>

Archived source code as at time of publication: DOI: [10.5281/zenodo.575664](https://doi.org/10.5281/zenodo.575664)

Software license: GPL-3

The data presented can be found at <https://github.com/rorynolan/exampletestr/tree/master/analysis>, along with a file showing how to reproduce that data.

Author contributions

Rory Nolan conceived the idea, created the package and wrote the paper. Sergi Padilla-Parra supervised the coding of the package, helped to test it and helped to write the paper.

Competing interests

No competing interests were disclosed.

Grant information

This work was been supported by the Wellcome Trust [105278] and [203141].

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

1. Wickham H: **R Packages**. O'Reilly, 2015. [Reference Source](#)
2. Wickham H: **testthat: Get started with testing**. *R J*. 2011; 3(1): 5–10. [Reference Source](#)
3. Hester J: **covr: Test Coverage for Packages**. R package version 2.2.2. 2017. [Reference Source](#)
4. Wickham H: **stringr: Simple, Consistent Wrappers for Common String Operations**. R package version 1.2.0. 2017. [Reference Source](#)

Open Peer Review

Current Referee Status:  

Version 2

Referee Report 29 June 2017

doi:[10.21956/wellcomeopenres.12933.r23685](https://doi.org/10.21956/wellcomeopenres.12933.r23685)



Laurent Gatto 

Computational Proteomics Unit, University of Cambridge, Cambridge, UK

Thank you for the amendments, Rory. I am happy to accept the publication.

Competing Interests: No competing interests were disclosed.

Referee Expertise: Computational biology, proteomics, data science, research software development, reproducible research.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Referee Report 12 June 2017

doi:[10.21956/wellcomeopenres.12567.r23062](https://doi.org/10.21956/wellcomeopenres.12567.r23062)



Laurent Gatto 

Computational Proteomics Unit, University of Cambridge, Cambridge, UK

The manuscript *Easier unit tests and better examples with examplestr and covr* describes the `examplestr` package that parses the example code in the manual pages and converts it into unit test stubs to be then completed by the developer.

I read the manuscript and found it was clearly written. I also successfully executed the examples in the package vignette. In general, I believe the authors have prepared a useful package and have described it well.

I have however two important points I would like to raise.

My first point is that I think it would be useful for the authors to provide a better rationale for unit testing. In my opinion, the current version of the text is obvious to those that already know about and practice unit testing. However, the goal of this work is to appeal to the many that haven't adopted unit testing in their

standard development yet. For those, it might be useful to provide more background and to emphasise the important of unit testing.

My second point is that by focusing on code from examples, some of the most important unit tests are systematically missed out. Indeed, one generally documents a typical, default, *best case* application of the code/package in the documentation files. One could argue that the goal of unit testing is to go beyond these, and assess the execution of the edge cases. Using some material from my [own work](#), some example would be

- to test the validity of empty data/objects (where users wouldn't, generally, produce empty data instances, and this wouldn't be described in the manual page);
- verify that an example object that is distributed with the package (and loaded with `load`), that was created from a distributed example files (for instance in `inst/extdata/`), remained valid and identical throughout the evolution of the classes and constructors;
- other examples illustrate how unit tests can be build and refined against a set of functions specifications, i.e. expected outputs.

Similarly, code presented in examples illustrate exported and high level code, while unit tests might rather want to address specific and private functions, that in turn are called within the high level user space functions.

Another important point is that writing (or thinking about) unit tests often influences how one writes code, testable code, which is not reflected in writing code in the manual examples.

Also, although efficiency (in terms of time when checking the package) is not a concern, at least in a first instance, a side effect of basing unit test on examples, results in re-running the same code twice. For unit testing, it might be worth wile to extend code coverage to rarely used lines of code by tuning arguments accordingly.

In the vignette, the author clearly states that "The Goal is NOT Fully Automated Unit Test Creation". May be this could also be mentioned in the manuscript.

A minor suggestion would be to drop `covr` from the title, as it might mislead readers that the manuscript also addresses directly that package.

Also, I was wondering what happened if an example code chunk was not run (wrapped in ``dontrun``). Is it ignores by `examplestr`?

To conclude, I think the manuscript would benefit greatly from a better introduction/discussion for newcomers. The second issue I have relates to the scope or validity of basing unit test on examples. While I do not expect `examplestr` (or any other package) to come up with valid general unit tests generation recipe, I think it would be essential to recast and discuss unit testing into a broader scope.

Is the rationale for developing the new software tool clearly explained?

Partly

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Partly

Competing Interests: No competing interests were disclosed.

Referee Expertise: Computational biology, proteomics, data science, research software development, reproducible research.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 13 Jun 2017

Rory Nolan,

Hi Laurent,

Thanks for the review and advice. I agree on all points and I'll edit the manuscript accordingly. I'll submit my revisions within the next 2 weeks.

For now, to answer your question, `\dontrun{}` blocks are ignored. I considered putting them into `expect_error()` statements but decided against it. I thought there was no good way to systematically deal with them. If you have a suggestion, please let me know.

I've edited the package documentation to make it clearer what happens to `\dontrun{}` blocks.

Thanks again,

Rory

Competing Interests: No competing interests were disclosed.

Author Response 16 Jun 2017

Rory Nolan,

Dear Laurent,

I have submitted a new version of the manuscript. I hereby outline my response to your comments.

I think it would be useful for the authors to provide a better rationale for unit testing. In my opinion, the current version of the text is obvious to those that already know about and practice unit testing. However, the goal of this work is to appeal to the many that haven't adopted unit testing in their standard development yet. For those, it might be useful to provide more background and to emphasise the importance of unit testing.

I have done this now in the introduction.

By focusing on code from examples, some of the most important unit tests are systematically missed out. Indeed, one generally documents a typical, default, best case application of the code/package in the documentation files. One could argue that the goal of unit testing is to go beyond these, and assess the execution of the edge cases.

I have included this point in the discussion.

Writing (or thinking about) unit tests often influences how one writes code, testable code, which is not reflected in writing code in the manual examples.

This is a good point. `exampletestr` is made for packages in which the author did not write tests as the package was being developed. In the manuscript I now try to make it clear that this isn't necessarily a good way to develop a tested package, but that it is the way that most packages are developed and hence `exampletestr` is a useful tool to begin testing these packages.

Although efficiency (in terms of time when checking the package) is not a concern, at least in a first instance, a side effect of basing unit test on examples, results in re-running the same code twice.

This is unfortunate, however I think having a set of unit tests which affirm that your examples are working properly very useful and worth having in spite of this re-running (which is, as you eluded to, not a big concern most of the time).

For unit testing, it might be worth while to extend code coverage to rarely used lines of code by tuning arguments accordingly.

This is an important point. I think this is taken care of by the workflow that makes use of `covr`.

In the vignette, the author clearly states that "The Goal is NOT Fully Automated Unit Test Creation". May be this could also be mentioned in the manuscript.

I decided not to copy this in explicitly because I think the discussion now makes clear the point that there is much more to unit testing than what `exampletestr` can do for you.

A minor suggestion would be to drop ``covr`` from the title, as it might mislead readers that the manuscript also addresses directly that package.

I have done this. Please let me know if you think the new title could be improved.

Also, I was wondering what happened if an example code chunk was not run (wrapped in ``\dontrun``). Is it ignored by ``exampletestr``?

`\dontrun{}` blocks are ignored. I considered putting them into `expect_error()` statements but decided against it. I thought there was no good way to systematically deal with them. If you have a suggestion, please let me know.

To conclude, I think the manuscript would benefit greatly from a better introduction/discussion for newcomers. The second issue I have relates to the scope or validity of basing unit test on examples. While I do not expect ``exampletestr`` (or any other package) to come up with valid general unit tests generation recipe, I think it would be essential to recast and discuss unit testing into a broader scope.

I have tried to address all of this. Thank you for your insightful review.

Rory

Competing Interests: No competing interests were disclosed.

Referee Report 30 May 2017

doi:[10.21956/wellcomeopenres.12567.r22849](https://doi.org/10.21956/wellcomeopenres.12567.r22849)



Thomas Quinn 

Bioinformatics Core Research Group, Deakin University, Geelong, Vic, Australia

The authors clearly outline the importance of unit testing. They then introduce a new package aimed to make it easier for developers to add unit tests to their own package. They do this by drawing from examples provided in the .Rd documentation. As such, the `exampletestr` package is automatically compatible with `Roxygen2`. Importantly, the authors outline unit tests using the `testthat` package, the standard unit testing package. These features make `exampletestr` consistent with best practices for R package development.

I used `exampletestr::make_tests_shells_pkg()` on an old deprecated package that I am renovating for the newest version of R. At first, I encountered an error; however, the package provided precise instructions on how to resolve the error. Otherwise, the `exampletestr` package works exactly as advertised. In my opinion, `exampletestr` would work nicely in combination with the `Rd2roxygen` package to help bring antiquated R packages up to a more rigorous and maintainable software standard.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 16 Jun 2017

Rory Nolan,

Dear Thomas,

I have submitted a revised version of the article to address Laurent Gatto's review.

This new version puts the package into better context.

The package has been updated such that the error that you got doesn't happen any more.

Thanks,

Rory

Competing Interests: No competing interests were disclosed.