A BRIEF INTRODUCTION TO PACKAGE TESTING WITH trackeR

Project Report

By

Riyaz Panjwani

Submitted in fulfillment of the requirements
Of
Summer Internship 2016
Undergraduate (B.Tech - sophomore)

Date 10th July 2016

ABSTRACT

Software testing[i] for production[ii] & application systems is paramount for package developers. This is because, the quality of coded package(s) has to be constantly monitored and modified to match user requirements. One of the reasons why R is so successful is a due to a huge variety of packages. Package testing, being one of the components of package development, ensures that performance metrics such as Scalability, Reliability and Resource usage are desirable. The goal of underling study is to develop an efficient tool for the users of R language to measure “Quantitative Metrics” such as Scalability of the Target package(s).There has been developments recently such as testthat[iii] package, Microbenchmark[iv] package, Pryr [v]Package and RUnit which provide tools for Package testing in R. The problem with the existing package-testing frameworks is that it cannot be used to track the performance of Target package directly parallel to development of functionality of Target package and some old package(s) do not use Unit test framework, testthat package. The project aims at developing a framework for R package developers which provides “Quantitative Metrics” such as scalability of Target package(s) over time for different Github commits along different Github Branches. The basic motivation is to introduce scalability testing which integrates with the workflow. By employing such a methodology the R Users can understand better about the Hardware performance of R Code (Wickam)[vi] [H]and can develop reliable packages with optimized code.

The development of trackeR package is achieved by, integration with github by using git2r package and accessing codes pertaining to last commits. Then performing run-time analysis the obtained code using microbenchmarking. The result obtained is a data-frame of commits attributes and its run-time

measurement. The result can be visualized by a plot, using ggplot2 package. As an example test, the stringr package is tested against its last 2 commits and graph is plotted to obtain the average increase in efficiency.

TABLE OF CONTENTS

Chapter Page

CHAPTER I: Introduction

The vitality and the need of Software performance testing can hardly be overrated[vii]. R[viii] begin an interpreted language emphasis over Software performance testing is apt. This is because, for compiled language, Compiler checks for the basic consistency of program during compilation. Though testing is important, it is tedious and quite frustrating from the developer's point of view, "testing is taken as a burden by the programmer than a boon" (Wickam, Advanced R)[H]. Therefore it is important to provide tools and framework which are easy-to-use, for all the users of R language to keep a track of performance metrics of target packages in quick way. The key goal of trakeR package is to help the developers and contributors know about how the efficiency of target package is increasing in regard to run-time. It aims at generating validating results which can be used for visualization and helps to predict the changes in the performance metric[ix] over time and issue a warning if the quality of code decreases[x] .

Over time there has been effort put in Software testing in R such as the development of testthat package (Wickam, R packages, 2014)[J], RUnit and svUnit[xi]. Experience shows that it is more feasible for package development on "code-test-simplify-improve" cycle than futile attempt to add test cases at the end of a large integrated complex software (Thomas Konig, November 5, 2015)[xii] [F]. Nevertheless the above does not provide any testing tool for code-runtime visualization or integration with GitHub & Git – A popular VCS. These considerations lead to various requirements that a useful –reliable framework should satisfy:

- The results should be easy to visualize.
- It should be possible to integrate with GitHub & Git.
- The results should be consistent with the time-line of development.
- It should give an idea of overall performance metrics over various changes accessible through an organized protocol.

It is standard practice to test commercial software at both unit and system level. In other words tests

are written for both individual components of the software and for the software as a whole. Through

CI[xiii] (Travis CI) results in rebuild of package and re-run of any unit test case. There has been many

programming languages like Smalltalk, Python, Java and Cpp which have the above

features[xiv].Nevertheless the package is built on to satisfy the above requirements utilizing the

developments made on similar grounds in a way to compliment them:

- Test results are displayed along with the summary of Commits from latest commits onwards
- Ease to export data for visualization.

The methods and approach used for making testthat package is highly useful to add on more

features to meet the above needs. Since it is comprehensively developed on account of various

similar experiences like JUnit and others (Wickam, testthat: Get Started with Testing, June 2011)[J].

- Create an automated system: Work over all the latest commits update when the cloned Repository is updated with the upstream branch and always measure the performance metrics over the recent commits by default 20.
- If the Code throws an error: Report as NA instead of quitting the program so that other properly coded test cases can be measured with accuracy.
- Integrate with work flow of package development: Provide necessary warnings in the result data indicating fall of accuracy

- Generate a list of commits: Generate a list of commits with their components in the form of a frame consisting of attributes such as Message, Sha-1 Values, Summary, Time of Commit and authors.

- Code a generic function to test the performance of Coded block in absence of testthat directory of R package.

CHAPTER II: Background and Literature Review

As rightly said: “it’s not that we don’t test our code, it’s that we don’t store our tests so they can be re-run automatically” (Wickam, testthat: Get Started with Testing, June 2011)[J]. Software testing is not a recent development or a new component of R. There has been work done previously for example: testthat draws inspiration from the xUnit family of testing packages, Ruby testing libraries like rspec[xv] , testy[xvi] , bacon[xvii] and cucumber[xviii]. (Wickam, testthat: Get Started with Testing, June 2011)[J] and so does RUnit from JUnit , CppUnit and PerlUnit (Thomas Konig, November 5, 2015)[F]. All of the above provide excellent and reliable tools for testing packages . Then why is a new package necessary? This is because of the rapid development of Open Source Community and VCS to replace traditional and tidious Software develoment on SourgeForge and Codeplex[xix](Davey)[B] due to a variety of resaons.There has to be a way to intergrate the work flow (On GitHub) and Effiecncy testing of the modified code on account of various recent commits. This helps us to generate a “Code-test-Simplify –Improve” Cycle (Thomas Konig, November 5, 2015)[F].

There has been a proof-of-concept package developed by Tobly Dylan Hocking[xx] which comapares the result between 2 commits of “animit package” over range of GST times.

It is standard practice to test commercial software at both unit and system level. In other words tests are written for the both the individual components of the software and for the software as a whole. Through Continuous Integration (CI) any change to the source code results in a rebuild of the package and re-run of any unit tests. This is essentially what happens when a package is submitted to CRAN (Galili, November 4th 2014)[C]. However, there is currently no easy way to see if package developers are altering the speed of their code. Likewise, there is no easy way to see if the disk/memory usage of functions defined in a package is changing over time. (Akash Tandon, 18th March 2015)[A].

As already mentioned this project is to make software testing embedded in workflow of the developers. Keeping in mind that most of the popular R Software including stringr (Wickam, Package 'stringr', April 30,2015)[I],testthat (Wickam, testthat: Get Started with Testing, June 2011)[J],testthatQuantity are all developed on GitHub. It becomes at most important to understand the structure of Git (Straub)[E]. When we commit changes in a repository we create a commit object. The Commit is addressable via hash (SHA-1 checksum – Secure Hash Algorithm)[xxi] which is calculated based on various other parameters of the commit object as its message, date, history and others. Commit object points to files (Stored as blob objects) via Tree object which are refereed by their SHA-1 Hash[xxii]. SHA-1 are generally 40 digits (20 Bytes)[xxiii] long and are rendered hexadecimal (Vogel, 10/08/2015)[G].

1. Package Testing

Package testing is of utmost importance especially in R since it is an interpreted language. It gives a sense of reliability of package and making efficient software for Real time analysis. Nevertheless package testing is very monotonous and frustrating task to perform. Adding to the above difficulties R being a slow language makes it necessary for developers of R language to implement “The Optimized Code” instead of “Seem-Appropriate-Code” hence the testing framework should be reliable. There are basically 2 main types of testing:

- Unit Testing
- Testing through Continuous Integration

The testing is required to be automated since we do not want to re-fix the fixed bugs again!

R basically is a functional programming language[xxiv],”Everything in R is either functions or Objects” (Wickam, Advanced R)[H].The architecuture of a programming language is bestdescribed by its paradigm. Paradigm is a way of classifing programming language accourding to the style of computer programming[xxv] .It is not necessary that a programming language may have only one paradigm like C++ , R is also a mutli-paradigm[xxvi] language.R has a strange feature in its architecture , that is, For computationally intensive tasks, C,C++, and Fortran code can be linked and called at run time (Hornik, 2016-06-06)[D]. Advanced users can write C, C++,Java, .NET or Python code to manipulate R objects directly.

CHAPTER III: Architecture of R Language and Modification

R has no formal definition nor does it have a well-defined architecture However for the sake of visualization we have Fig 3.1 which is an attempt to show the existing architecture and the modifications made by the trackeR package – Testing framework.

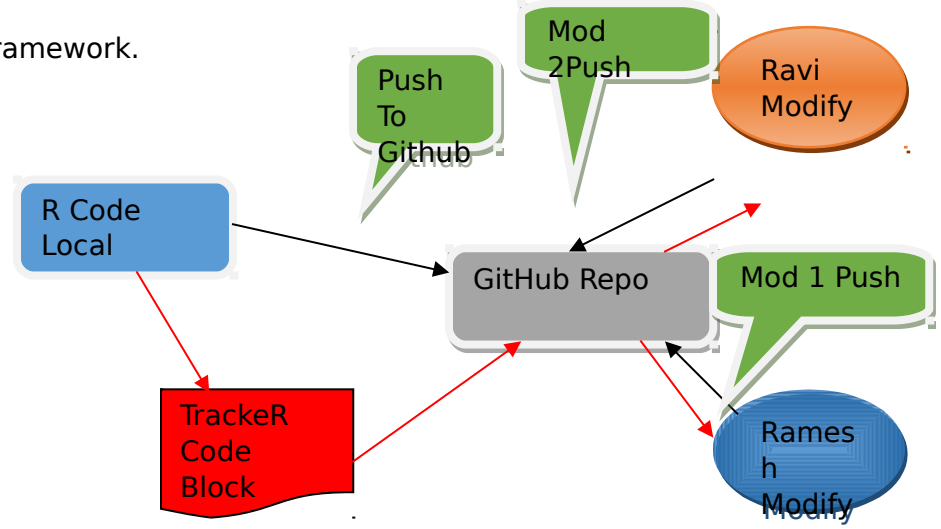


Fig 3.1 Architecture of GitHub and the Red Block and arrows indicated the trackeR package flow

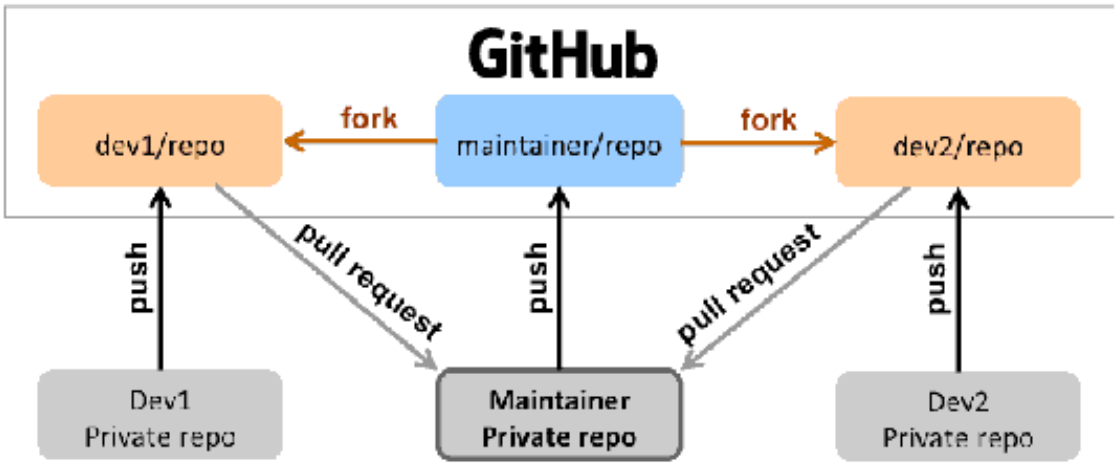
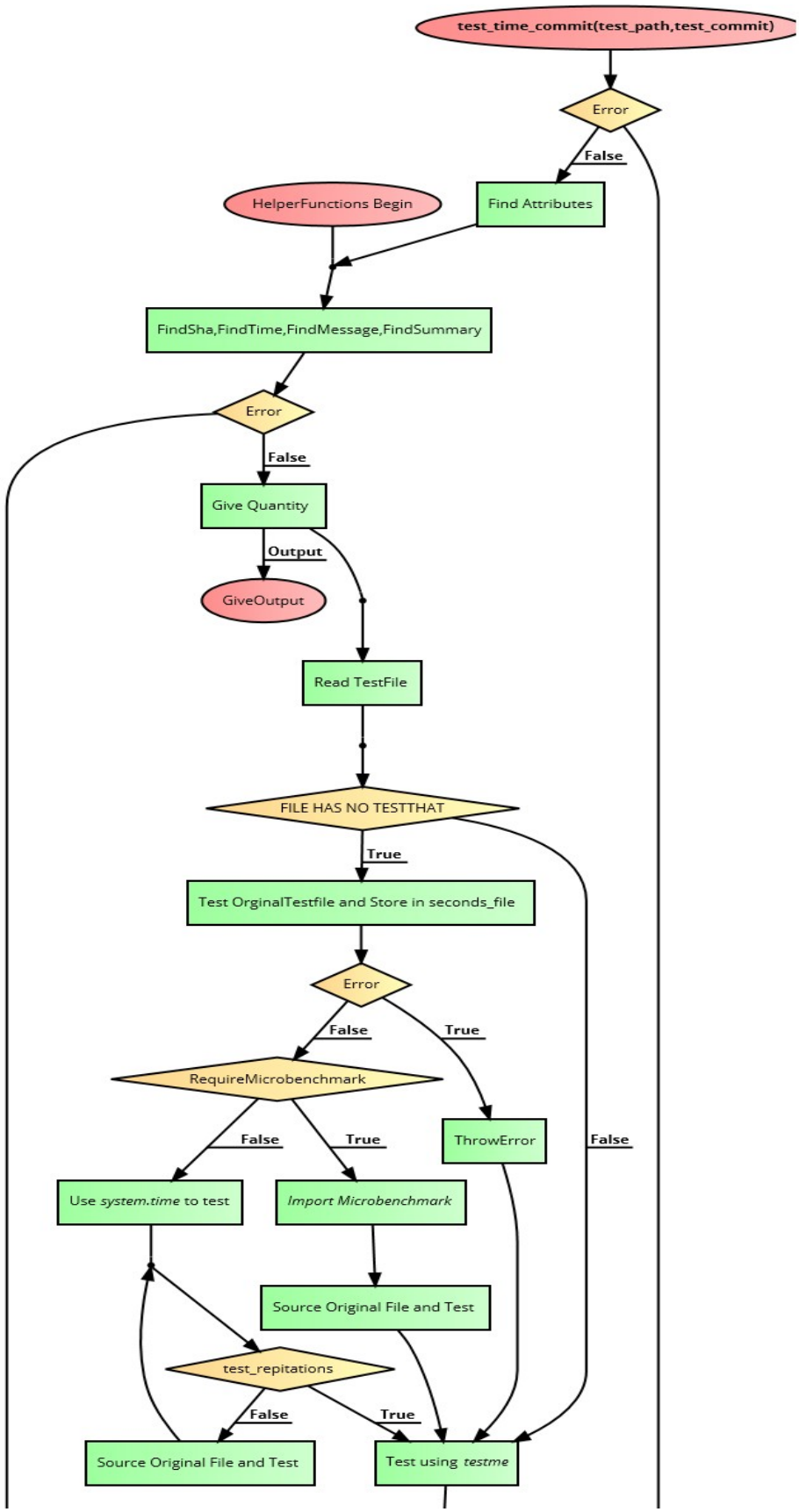


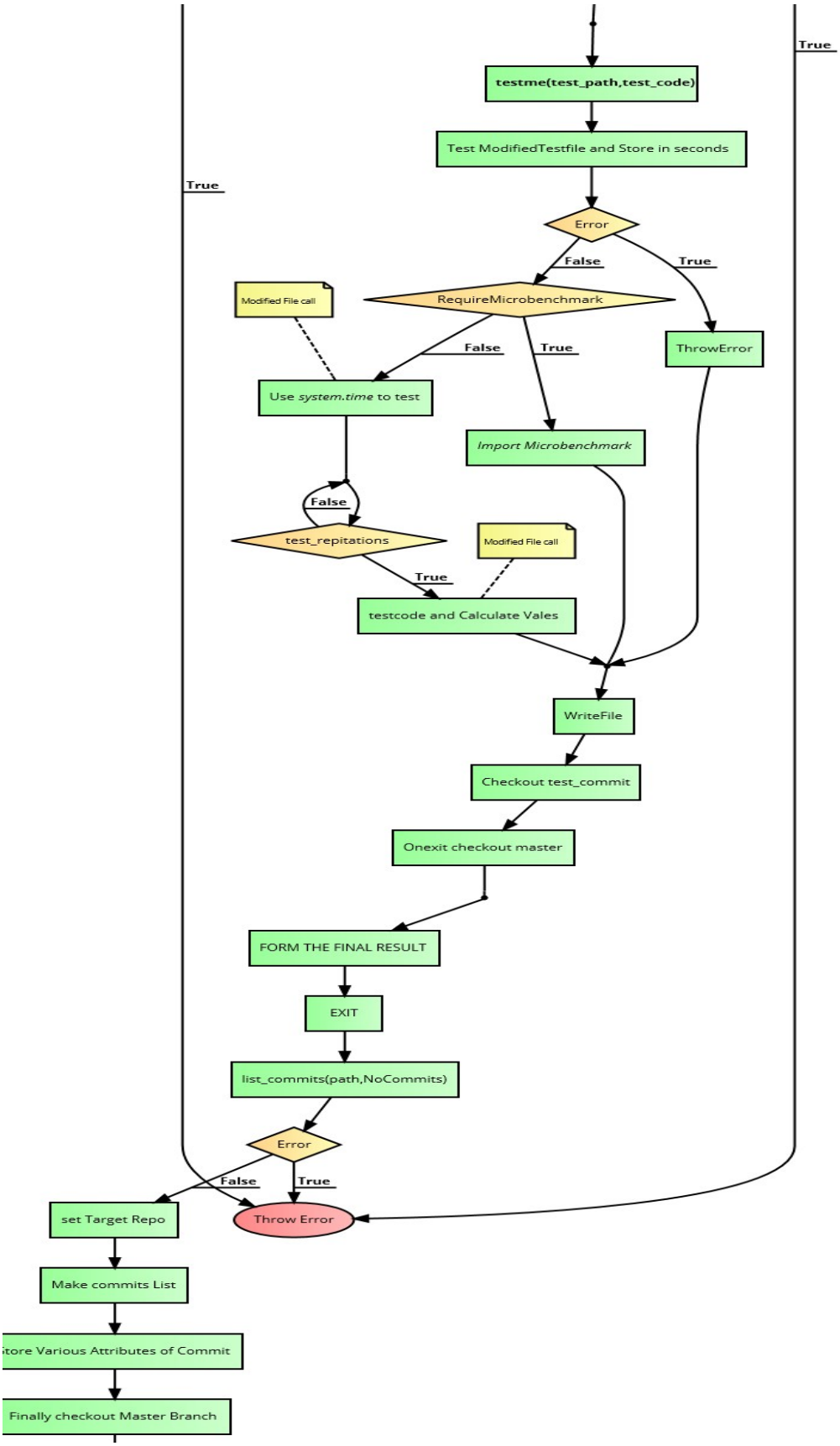
Fig 3.2 A Diagramatic representation of Structure of Github

The above are simple diagrammatic way of explaining the working of Git and the integration of trackeR package to make the study more appropriate to the given context we shall introduce a Functional diagram of ***test_time_commits***() functions which is the heart of the package and later refer this chart to explain its working. The chart is a simple explanation to how the project works at its core it has mainly the flowing functions which it calls:

- Helper Functions for getting SHA Values, Message , Summary and Time of Commit
- List_commits() which gives us list of commits for args as number of commits and path/to/repo
- Test_me() as a way to replace the traditional testthat pieces of code if present or else directly use the code to test for its efficiency

Fig 3.3 Flow diagram of trackeR package


test_time_commit(test_path,test_commit)
Error
False
HelperFunctions Begin
Find Attributes
FindSha,FindTime,FindMessage,FindSummary
Error
False
Give Quantity
Output
GiveOutput
Read TestFile
FILE HAS NO TESTTHAT
True
Test OrginalTestfile and Store in seconds_file
Error
False
True
RequireMicrobenchmark
False
True
ThrowError
False
Use system.time to test
Import Microbenchmark
Source Original File and Test
test_repitations
False
True
Source Original File and Test
Test using testme


True
testme(test_path,test_code)
Test ModifiedTestfile and Store in seconds
True
Error
False
True
Modified File call
RequireMicrobenchmark
False
True
ThrowError
Use system.time to test
Import Microbenchmark
False
test_repitations
Modified File call
True
testcode and Calculate Vales
WriteFile
Checkout test_commit
Onexit checkout master
FORM THE FINAL RESULT
EXIT
list_commits(path,NoCommits)
Error
False
True
set Target Repo
Throw Error
Make commits List
tore Various Attributes of Commit
Finally checkout Master Branch

CHAPTER IV: METHODOLOGY AND ALGORITHMS

This chapter aims at navigating through the development of framework for testing. We basically divide our working in 2 parts:

- Connecting with GitHub to extract the details of Code and Commits
- Using the the obtained code to test for its speed using Microbenchmarking / Sytem Time

4.1 Extracting Commit Details From GitHub :

The function responsible for this objective is list_commits() in trackeR package. The basic idea of this component is to extract commit details aka commit attributes which is the Meta-data that defines the commit like SHA1 Value, Message, Summary, Time of Modification and Author of Modification. This forms the structure of a commit[xxvii].

NOTE: The trackeR package requires you to set the current directory to the concerned git repository before using the functions.

Usage :

list_commits(path-to-repo,number_of_commits)

The function has 2 arguments path-to-repo: This represents the “path/to/repo” which links the path to cloned repository if you are not the owner of repository and the other lets you decide number of recent commits to be enlisted.

Output: The table outputs a data-frame with the above mentioned attributes (See Table 4.1,Table 4.2, Table 4.3).

ALGO & Results:

```
##Gives the List of Commits given path and number of commits
#Data frame stores the SHA1 Values
Funtion list_commits start:
  Step1: Take input path/to/repo and Number of Commits
  Step2: Check for Error Conditions
  Step3: Create a Variable target and store meta-data of repository
  Step4: Create a list of Commits in target repo specified by
num_of_commits
  Step5: do I in num_of_commits
              Store the data pertaining to various attributes
         End loop;
  Step6: Checkout to master branch
  Step7: Bind the results in a data frame
Function list_commit end;
```

4.1.1 TABLE OF LIST OF SUMMARY OF 10 COMMITS OF STRINGR PACKAGE

Comment Numbers	Summary
1	Add simplify arg to str_split
2	News tweaking
3	Style tweaks to string interpolation.
4	Tweaking test style
5	Convert to modern testhat format
6	Implement str_trunc.
7	Merge branch 'master' of github.com:hadley/stringr
8	Merge pull request #57 from craigcitro/partials
9	Fix existing partial matches in stringr.
10	Merge pull request #101 from lmullen/fix-sentence-splitting

4.1.2 TABLE OF LIST OF MESSAGES OF 10 COMMITS OF STRINGR PACKAGE

Comment Numbers	Message
1	Add simplify arg to str_split\n
2	News tweaking\n
3	Style tweaks to string interpolation.\n\nRemove formula interface in favour of vector.\n
4	Tweaking test style\n
5	Convert to modern testhat format\n
6	Implement str_trunc.\n\nFixes #65.\n
7	Merge branch 'master' of github.com:hadley/stringr\n
8	Merge pull request #57 from craigcitro/partials\n\nFix existing partial
9	Fix existing partial matches in stringr.\n
10	Merge pull request #101 from lmullen/fix-sentence-splitting\n\nFix sentence splitting

4.1.3 TABLE OF LIST OF DATE OF COMMIT AND SHA VALUES OF 10 COMMITS OF STRINGR PACKAGE

Comment Numbers	Date Of Commit SHA VALUE
1	2015-11-05 07:09:01 a67f8f06d4c8e8cac98d3361af66cc209b33f446
2	2015-11-04 08:49:00 b5720ec7804bfcbee30c472dc0ec2a359313de8e
3	2015-11-02 11:03:08 0092b930465ba91af62858578c940623e24977fb
4	2015-11-02 10:53:08 e8d5ef3e30fd8e1df8aecbdc00f040bd583b18eb
5	2015-11-02 10:51:25 96d74409bfecb5dc18a8ab426c67cb19767e386f
6	2015-11-02 07:58:33 25180cac6ca27255f960ef15d04f57a87c4e54f4
7	2015-11-01 20:58:59 c26f19ab323e49d61f7adc16f971e591a9477eb4
8	2015-10-31 08:07:22 81563396a51a1ba443e25f41d0c4aa7224180c82
9	2015-10-30 21:19:40 b09887d10ec9d88fce8ae9523ab7241262fb88ca
10	2015-10-30 16:53:57 c7a94d0743b6c48664711cd3ed8b4fc135c48052

4.2 Test the Efficiency of code :

This is a must-need tool that helps you get the efficiency in terms of Runtime of your code. This can also be applied to some standard functions like Sum, sub setting, Apply family, loops and others.

Usage:

testme(Test Name, Code)

The output is a data frame which enlists the runtime and the test run on it by-default this is set to 3 test reputations which is custom to the user. The user can define his number of repetitions by modifying test.repetitions in his code.

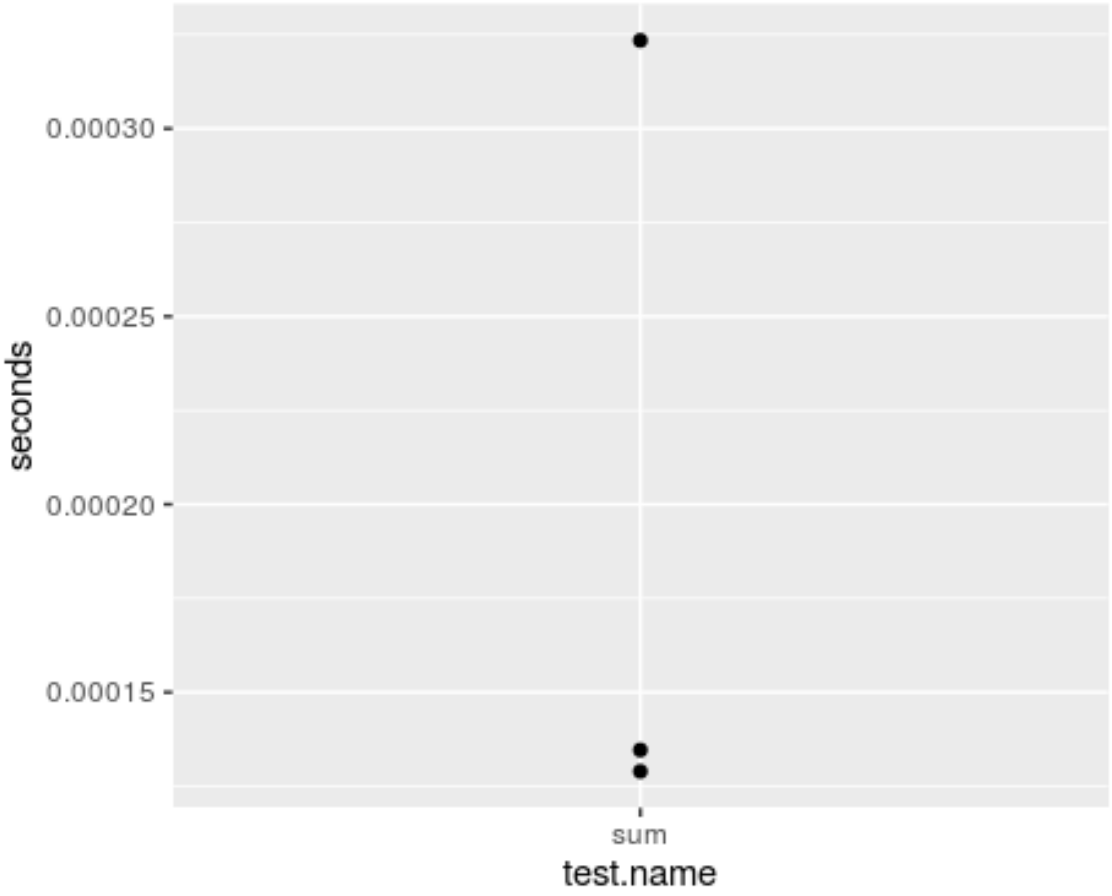
There can be a better visualization via graph which can be embedded via the ggplot2 package which can be imported and then the graph is displayed in the “plot” section of RStudio.

NOTE:

The Functions described above work free of bugs in RStudio however other IDE’s like RKWard may report bugs due to installation issues.

Test Results:

Fig 4.2.1 PLOT OF STANDARD SUM FUNCTION RUN TIME ANALYSIS



4.1.3 TABLE OF RESULTS GENERATED FOR RUN-TIME ANALYSIS OF SUM FUNCTION

Test.name	Seconds (in ns)
sum	0.0002962362
sum	0.0001415123
sum	0.000164242

The above data frame is displayed on the console. A detailed description of algorithm used is given below.

ALGO:

```
Step1: Initialize: empty list of test_results
Step2: Initialize: Number of test_repitaions
Function testme start:
  Step3: Take input test_name and Code
  Step4: Create an environment
  Step5: Obtained the exact code passed in a variable
  IF Microbenchmark package is present:
  Step6: Test the code for various cases and obtain the results
  Else
  Step7: Use system.time to test for various cases
  Step8: Store the results in a data-frame
Function testme end;
```

4.3 HELPER GITHUB :

This component is not so particularly interesting as it derives from the Sec4.1 except in a sense that the programmer may not require the list_commits() to his advantage as it enlists all the commits so in order to access particular commits via the commit Number you may have to use github helper functions which given the Commit Number gives the required attribute values. The find_(attribute)

Gives the required attribute of the package.

NOTE: The trackeR package requires you to set the current directory to the concerned git repository before using the functions.

Usage:

find_{attribute}(val_commit)

Output: gives the commit attribute

ALGO FOR HELPER FUNCTIONS:

```
##Helper Functions
Funtion Find_sha start:
  Step1: Input val_of_commit
  Step2: Check if error :
     Step3: Throw error statement
Else
     Step3: Obtain the sha value by val_commit
Funtion Find_sha end;

Funtion Find_time start:
  Step1: Input val_of_commit
  Step2: Check if error:
     Step3: Throw error statement
Else
     Step3: Obtain the time value by val_commit
Funtion Find_time end;

Funtion Find_msg start:
  Step1: Input val_of_commit
  Step2: Check if error:
     Step3: Throw error statement
Else
     Step3: Obtain the msg by val_commit
Funtion Find_msg end;

Funtion Find_Summary start:
  Step1: Input val_of_commit
  Step2: Check if error:
     Step3: Throw error statement
Else
     Step3: Obtain the Summary by val_commit
Funtion Find_Summary end;
```

4.4 THE TEST TIME COMMIT :

This component takes accesses all possible commits specified by number_of_commits and tests its code run-time which it then stores in a data frame with the commit attributes.

Usage: test_time_commit (test_path, num_commits)

Output: data frame with commit -> attribute – Runtime format

There are some error handling functions included one such is stopifnot(<condition>) which throws error on failing. This function combined with helper and testme functions help in forming a complete framework. The results can be plotted and investigated to give the complete output.

ALGO:

```
Funtion test_time_commit start:
  Step1: Take input test_path, test_commit
  Step2: Check for error
  Step3: Get the structure of test_commit
  Step4: Read code from test_path
  Step5: Maintain a temp_org file to original code
  Step6: Maintain another temp_mod file incase the package doesnot use
testthat package
   Step7: Modify the code in temp_mod by replacing “testthat(” with “testme(”
   Step8: Close connection
Funtion test_time_commit end;
```

4.5 THE WRAPPER :

All the components are nicely wrapped in the wrapper R script File which has test_time_commit as its main functionality. The following is the flow of how the functions link and operate to give the desired objective:

- Give the test_path which is the path in repository which stores the tests mostly it has the name tests/{ Particular tests (Eg: string-split.r of stringr package) }
- Go to the github Repository and access the particular commits and the code
- Test the Runtime against various commits
- Store the run time against various attributes in a data-frame
- Plot the graph against the obtained data frame

Fig 4.4.1 FINAL PLOT

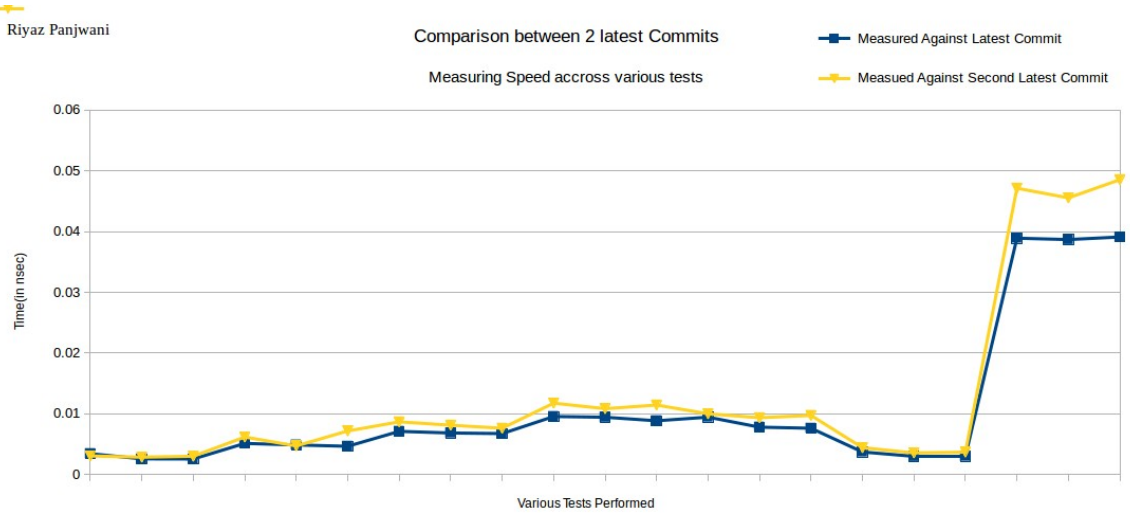


Fig 4.5.1 OUTPUT DATA FRAME

Sno	test.name	MeasuredParam	status	MeasuredVal	message	date_time	summary
1	special cases are correct	seconds	pass	0.002460241	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
2	special cases are correct	seconds	pass	0.0019630 3	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
3	special cases are correct	seconds	pass	0.002013316	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
4	str_split functions as expected	seconds	pass	0.003977732	Merge pull request #101 from lmullen/fix-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-

					sentence-		sentence-
5	str_split functions as expected	seconds	pass	0.003527088	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
6	str_split functions as expected	seconds	pass	0.003463335	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
7	vectors give correct results dealt with correctly	seconds	pass	0.005925106	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
8	vectors give correct results dealt with correctly	seconds	pass	0.005391278	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
9	vectors give correct results dealt with correctly	seconds	pass	0.005417082	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
10	n sets maximum number of splits in str_split	seconds	pass	0.007493963	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
11	n sets maximum number of splits in str_split	seconds	pass	0.006950006	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
12	n sets maximum number of splits in str_split	seconds	pass	0.007036318	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
13	n sets exact number of splits in	seconds	pass	0.006428	Merge pull request #101 from	2015-10-30 16:53	Merge pull request #101 from

	str_split_fixed			43	lmullen/fix-sentence-	:57	lmullen/fix-sentence-
14	n sets exact number of splits in str_split_fixed	seconds	pass	0.000603021 9	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
15	n sets exact number of splits in str_split_fixed	seconds	pass	0.000600305 9	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
16	str_split can split sentences correctly	seconds	pass	0.002898025	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
17	str_split can split sentences correctly	seconds	pass	0.002324981	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
18	str_split can split sentences correctly	seconds	pass	0.002382 38	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
19	test-split.r	seconds	Success	0.030361153	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
20	test-split.r	seconds	Success	0.029784752	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-
21	test-split.r	seconds	Success	0.030149435	Merge pull request #101 from lmullen/fix-sentence-	2015-10-30 16:53:57	Merge pull request #101 from lmullen/fix-sentence-

CHAPTER IV: RESULTS

The plot (see Fig 4.4.1) gives an insight to the various runtimes against tests of stringr packageabove. As it can be seen that the efficiency is increasing over the past commits (The Blue line is Below Yellow line which means run time is decreasing and hence efficiency is increasing). This means that the latest commit is an optimized code and hence the commit made is time efficient.

The data frame (See Table 4.5.1) gives a Quantitative estimation of Run time analysis. Looking at the last 2 test case of the recent and last 2nd commit it can be inferred from the graph that the Runtime is decreased by 28.3% (Second Recent: 0.0511 ns and Recent: 0.398 ns) and the overall average being 30.2% which means that the efficiency increases and hence the modified code is better (when compared in regard to run-time efficiency)

CHAPTER V: CONCLUSION

The framework succeeds in providing a quantitative as well as qualitative measurement of code run-time analysis for the various tests of a package and we

have connected to GitHub in order to access the commits which in turn helps us access the code in the past. Thus the aim at developing a handy framework is done and we have also measured the amount by which our code gets better.

References

[A]Akash Tandon, T. D. (18th March 2015). *Test timings on Travis.*

[B]Davey, M. (n.d.). Why is GitHub so Popular ?

[C]Galili, T. (November 4th 2014). Analyzing coverage of R unit tests in packages – the {testCoverage} package. *Analyzing coverage of R unit tests in packages – the {testCoverage} package*, 5-20.

[D]Hornik, K. (2016-06-06). R FAQ. *Frequently Asked Questions on R*.

[E]Straub, S. C. (n.d.). *ProGit.* Mountain view,CA: Apress.

[F]Thomas Konig, K. J. (November 5, 2015). RUnit. *RUnit - A Unit Test Framework for R*, 10-15.

[G]Vogel, L. (10/08/2015). Git Tutorial. *Git article* .

[H]Wickam, H. (n.d.). *Advanced R.*

[I]Wickam, H. (April 30,2015). Package 'stringr'. *Package* '*stringr*', 1-23.

[J]Wickam, H. (June 2011). testthat: Get Started with Testing. *testthat: Get Started with Testing*, 15-20.

Appendix A

Github is a web based Git repository hosting service. It offers all distributed revision control and SCM functionality of Git along with its features. The objects in Git are called Git Objects. When we commit changes in a repository we create a commit object. The Commit is addressable via hash (SHA-1 checksum – Secure Hash Algorithm) which is calculated based on various other parameters of the commit object as its message, date, history and others. Commit object points to files (Stored as blob objects) via Tree object which are refereed by their SHA-1 Hash. SHA-1 are generally 40 digits (20 Bytes) long and are rendered hexadecimal.

A typical example of internal representation of generating SHA-1 hash is shown below.

```
sha1(
    commit message  => "initial commit"
    commiter        => riyazpanjwani96
<riyaz.panjwani@gmail.com>
    commit date     => Fri July 1 10:56:57 2016 +0100
    author          => riyazpanjwani96
<riyaz.panjwani@gmail.com>
    author date     => Fri July 1 10:56:57 2016 +0100
    tree            =>
9c435a86e664be00db0d973e981425e4a3ef3f8d
)
```

Bibliography

[A]Akash Tandon, T. D. (18th March 2015). *Test timings on Travis.*

[B]Davey, M. (n.d.). Why is GitHub so Popular ?

[C]Galili, T. (November 4th 2014). Analyzing coverage of R unit tests in packages – the {testCoverage} package. *Analyzing coverage of R unit tests in packages – the {testCoverage} package*, 5-20.

[D]Hornik, K. (2016-06-06). R FAQ. *Frequently Asked Questions on R*.

[E]Straub, S. C. (n.d.). *ProGit.* Mountain view,CA: Apress.

[F]Thomas Konig, K. J. (November 5, 2015). RUnit. *RUnit - A Unit Test Framework for R*, 10-15.

[G]Vogel, L. (10/08/2015). Git Tutorial. *Git article* .

[H]Wickam, H. (n.d.). *Advanced R.*

[I]Wickam, H. (April 30,2015). Package 'stringr'. *Package* '*stringr*', 1-23.

[J]Wickam, H. (June 2011). testthat: Get Started with Testing. *testthat: Get Started with Testing*, 15-20.

[i] https://en.wikipedia.org/wiki/Software_testing

[ii] https://en.wikipedia.org/wiki/Production_system

[iii] https://cran.r-project.org/web/packages/testthat/index.html

[iv] https://cran.r-project.org/web/packages/microbenchmark/microbenchmark.pdf

[v]

[vi] *Hadley Wickam Advanced R*

[vii] Anatonia Bertolino ISTI-CNR di Pissa , Italy bertolino@iei.cnr.it

[viii] https://en.wikipedia.org/wiki/R_(programming_language)

[ix] Here it is Run time optimization only , Advanced features are to be added

[x] The Quality of Code can be measured on several factors, we shall only refer to run-time efficiency as the key parameter.

[xi] https://cran.r-project.org/web/packages/svUnit/index.html

[xii] Thomas K¨onig, Klaus Junemann, and Matthias Burger ¨ Epigenomics AG ***RUnit - A Unit Test Framework for R*** November 5, 2015

[xiii] Continuous Integration

[xiv] http://ronjeffries.com/xprog/testfram.html

[xv] http://rspec.info/

[xvi] https://github.com/ahoward/testy

[xvii] https://github.com/chneukirchen/bacon

[xviii] https://github.com/aslakhellesoy/cucumber/wiki/

[xix] Traditonally hosted Non-Distributed VCS

[xx] https://github.com/tdhock

[xxi] https://developer.github.com/v3/repos/contents/

[xxii] https://en.wikipedia.org/wiki/SHA-1

[xxiii] https://git-scm.com/book/en/v2/Git-Internals-Git-References

[xxiv] https://en.wikipedia.org/wiki/Functional_programming

[xxv] https://en.wikipedia.org/wiki/Computer_programming

[xxvi] Array , OO, Imperative , Functional and procedural

[xxvii] For more detail refer to the Appendix A