-Sustainable research software is defined as software that can be reused, in future projects.

- good coding practices will lead to more sustainable software

- it has been proposed that sustainability should be considered as a requirement of a project.

- This research aims measuring the relationship between characteristics of a project's codebase and the project's active life, which we term software sustainment.

- I collect data for this study from GitHub, selecting a subset of repositories based on their start date and the language they are written in.

-I take an empirical approach to understanding what makes a project sustainable, by examining how its code quality affect its sustainment.

-My definition of software sustainment is the time period from the initial creation of the software in a repository to the last commit in the original repository. where S is our software sustainment metric, measured in days.

- Projects were mined from GitHub have following criteria: they were created between 1st January and 31st December 2009; they were written in Java.

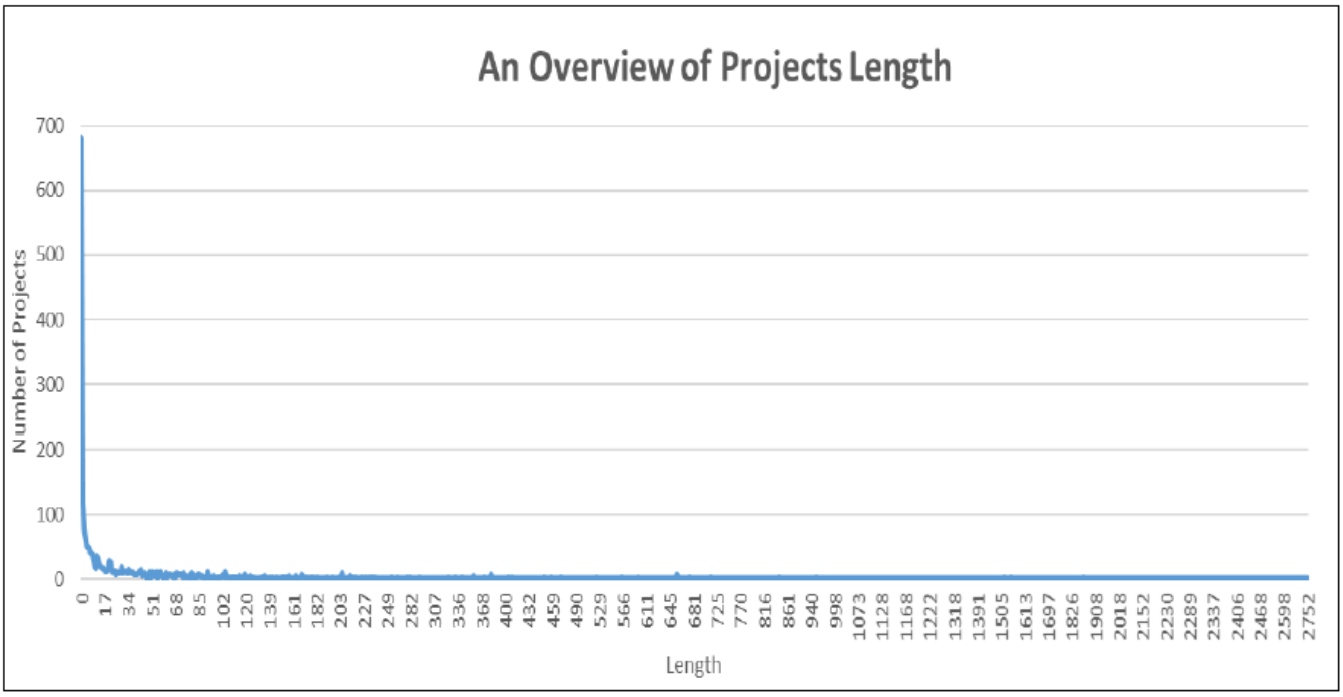


Figure 1 shows the distribution of projects as a function of S, in days. Of 3113 projects in total, 22% (682) had an S value of 0, and 35% (1076) had an S value < 7, indicating that over a third of the projects were sustained for only a week. A cursory inspection reveals some of these projects to be quite large, so it is likely that in these cases the development period was longer than the value we calculated according to our sustainment metric, and that the project was only put into Git version control sometime after its real start date. After

the steep drop off at around seven days, the curve gradually flattens over time.

- I used the following static analytic metrics to measure code quality of projects sample:

1-Lines of Code (LOC) is an indication of class size, where a higher value means longer and potentially more complex code. It is advisable to treat this metric in relative, rather than absolute terms, as lines of code may vary with programming language, or the individual style of a programmer.

2- Number of local Methods (NOM), an indicator of interface complexity, measures the number of methods locally declared in a class. As the interface grows, the class becomes more complex, and more difficult to test. The optimum value for this metric is generally considered to be between 3 and 7. If there are fewer than 3, the class might be simply a data holder; if there are more than 7, the class might be incoherent or in need of decomposition.

3-Depth of Inheritance Tree (DIT) was selected because inheritance is a basic yet powerful concept of object oriented languages. DIT calculates the complexity of a software entity based on the distance between a node and its root down the inheritance tree. As the code goes down the inheritance tree, testing becomes more difficult as the control flow becomes more complicated. A value between 0 and 4 is generally considered to indicate an adequate balance between complexity and the use of inheritance.

4-Coupling Between Objects (CBO) calculates the complexity of a class through its dependencies: a class is considered well designed when it is loosely coupled. Classes with a large number of dependencies are more difficult to maintain and test. The reusability of classes is limited by high levels of coupling because if a class depends on other classes, it is difficult to reuse it in another system. A value of this metric greater than 4 is generally considered undesirable because it indicates a high number of dependencies.

5-Improvement of Lack of Cohesion in Methods (ILCOM) provides a measure of class cohesion, by calculating the number of connected components in a class. High cohesion is a desirable characteristic within a class in object oriented languages, and this metric is a possible indicator of how well a class was designed. It is usually harder to test classes that do not have cohesion between their components. A value of zero in the ILCOM metric indicates a lack of methods in the class, while a value of one represents a high level of cohesion. A value greater than one indicates cohesion is low, and the class may benefit from being divided into separate classes.

6-Lack of Documentation (LOD) was chosen as an interesting metric that considers comments in the code, with at least one comment per method and one per class as a minimum target. Comments often make the purpose of methods and classes clearer, increasing maintainability and facilitating the reuse of the code. Comments in Java code can also be used to automatically build API documentation for a project, so one might expect well maintained code to include at least one comment per method and per class for this purpose. A caveat is that the content of the comments is not considered.

Here more explanation for these metrics

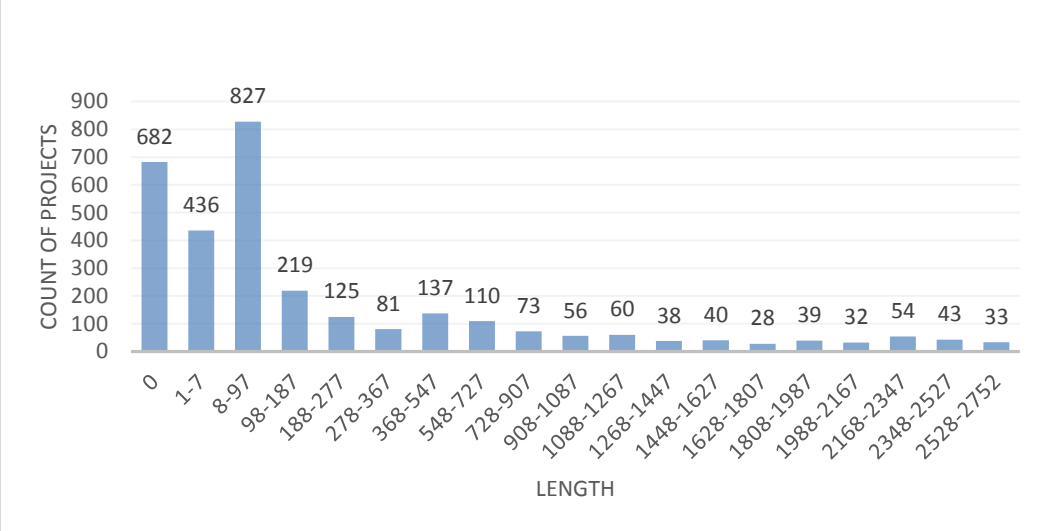
1- http://www.arisa.se/compendium/
2- http://support.objecteering.com/objecteering6.1/help/us/metrics/toc.htm
3-

- Since there are 3113 different projects and it is impossible to analyse all. I divided the time duration to 19 segments as following:

- 0 days
- 1-7 days
- Every 90 days for the rest of the first year.
- Every 180 days the rest duration.

#	Length	Count of Projects
1	0	682
2	1-7	436
3	8-97	827
4	98-187	219
5	188-277	125
6	278-367	81
7	368-547	137
8	548-727	110
9	728-907	73
10	908-1087	56
11	1088-1267	60
12	1268-1447	38
13	1448-1627	40
14	1628-1807	28
15	1808-1987	39
16	1988-2167	32
17	2168-2347	54
18	2348-2527	43
19	2528-2752	33
Grand Total		3113

-



- I selected 7 projects randomly from each segment and analyze it using scripts to get code quality metrics (look at Projects CQ analysis folder).
- then I got the average of each metrics for each project which is in file projects metrics.
- then I got the average of each segment by calculated the average of all projects in that segment and this is which in code quality analysis file.

The task is

Part 1 descriptive analysis for each metric

first, you have to do a descriptive analysis for each metric
like how I have done it about Sustainment matric
in code quality analysis file sheet2 have an example graph

you may scatter plot graph instead of a line graph

you have to describe in details and make some hypotheses according to result

Part 2 statistical analysis
you can use Python R or SPSS up to you
for example, you may do logistic regression between LOD and S

do a statistical analysis for each metrics and S
try to find a relationship

final work will be a report that have analysis + step to get the analysis

any scripts u used, and file (excel, Spss,..)

Be creative and use grphs , heatmap … etc to support your explaination.