What is the Truck Factor of Popular GitHub Applications? A First Assessment

Guilherme Avelino, Marco Tulio Valente, Andre Hora

Department of Computer Science, UFMG, Brazil

{gaa,mtov,hora}@dcc.ufmg.br

Abstract

The Truck Factor designates the minimal number of developers that have to be hit by a truck (or quit) before a project is incapacitated. It can be seen as a measurement of the concentration of information in individual team members. We calculate the Truck Factor for 133 popular GitHub applications, in six languages. Results show that most systems have a small truck factor (46% have Truck Factor=1 and 28% have Truck Factor=2).

1 Introduction

The *Truck Factor* designates the minimal number of developers that have to be hit by a truck (or quit) before a project is incapacitated [1]. The Wikipedia defines that it is a "*measurement of the concentration of information in individual team members. A high Truck Factor means that many individuals know enough to carry on and the project could still succeed even in very adverse events.*"[1] The term is also known by *Bus Factor/Number*.

In this paper, we report the first results of a study conducted to estimate the Truck Factor of popular GitHub applications. Our results show that most systems have a small truck factor (46% have Truck Factor=1 and 28% have Truck Factor=2). Section 2 reports our study setup, including a description of the technique we used to calculate code authorship, the dataset used in the paper, and the heuristic we used to estimate the Truck Factor. Section 3 presents our first results.

[1] `https://en.wikipedia.org/wiki/Bus_factor`

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015

2 Study Setup

2.1 Code Authorship

We define an *author* as a developer able to influence or command the implementation of a file. Therefore, she is not a collaborator with some expertise in the file, but for example someone who is able to lead other developers when working in the file. To define the authors of a file, we rely on the *Degree of Authorship* (DOA) measure [2, 3], which is is computed as follows:

$$DOA = 3.293 + 1.098 * FA + 0.164 * DL - 0.321 * \ln(1 + AC)$$

The degree of authorship of a developer d in a file f depends on three factors: first authorship (FA), number of deliveries (DL), and number of acceptances (AC). If d is the author of f, FA is 1; otherwise it is 0; DL is the number of changes in f made by D; and AC is the number of changes in f made by other developers. Basically, the weights of each variable assume that FA is by far the strongest predictor of file authorship. Recency information (DL) also contributes positively to authorship, but with less importance. Finally, changes by other developers (AC) contribute to decrease someone's DOA, but at a slower rate. The weights used in the DOA equation were empirically derived through an experiment with seven professional Java developers [2]. The authors also showed that the model is robust enough to be used in different environments and projects.

In this study we consider only *normalized DOA* values. For a file f, the normalized DOA ranges from 0 to 1, where 1 is granted to the developer with the highest absolute DOA among the developers that worked on f. A developer d is an author of a file f if its normalized DOA is greater than a threshold k. We assume $k = 0.75$, which is a value that presented reasonable accuracy in a manual validation we performed with a sample of systems.

2.2 Dataset

We evaluate systems implemented in the six languages with the largest number of repositories in GitHub: JavaScript, Python, Ruby, C/C++, Java, and PHP. We initially select the top-100 most popular systems in each language, regarding their number of stars (starring is a GitHub feature that lets users show their interest on repositories. Considering only the systems in a given language, we compute the first quartile of the distribution of three measures: number of developers, number of commits, and number of files (as collected from GitHub on February 25th, 2015). We then discard systems that are in the first quartiles of any of these measures. The goal is to focus on the most important systems per language, implemented

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015

by teams with a considerable number of active developers and with a considerable number of files. A similar procedure is followed by other studies on GitHub [4].

After this first selection, we remove repositories with evidences of being incorrectly migrated to GitHub (from another repository, like SVN). Specifically, we remove systems having more than 50% of their files added in less than 20 commits (*i.e.,* less than 10% of the minimal number of commits we initially considered). This is an evidence that the system was developed using another version control platform and the migration to GitHub did not preserve its previous version history. Finally, we manually inspected the GitHub page of the selected systems. As result, we decided to remove the repositories RASPBERRYPI/LINUX and DJANGO/DJANGO-OLD. The first is very similar to TORVALDS/LINUX and the second is an old version of a repository already in the dataset.

Table 1 summarizes the final list of repositories we selected for the study. It includes 133 systems, in six languages; Ruby is the language with more systems (33 systems) and PHP is the language with less systems (17 systems). Considering all systems, the dataset includes more than 373K files, 41 MLOC, and 2 million commits.

Table 1: Dataset

Language	**Repositories**	**Developers**	**Commits**	**Files**	**LOC**
JavaScript	22	5,740	108,080	24,688	3,661,722
Python	22	8,627	276,174	35,315	2,237,930
Ruby	33	19,960	307,603	33,556	2,612,503
C/C++	18	21,039	847,867	107,464	19,915,316
Java	21	4,499	418,003	140,871	10,672,918
PHP	17	3,329	125,626	31,221	2,215,972
Total	**133**	**63,194**	**2,083,353**	**373,115**	**41,316,361**

File Cleaning: Studies on code authorship should consider only files representing the source code of the selected systems. Therefore, files representing documentation, images, examples, etc should be discarded. Moreover, it is also fundamental to discard source files associated to third-party libraries, which are frequently found in repositories of systems implemented in dynamic languages. For this purpose, we initially used the Linguist library[2], which is the tool used by GitHub to show the percentage of files in a repository implemented in different programming languages. We excluded from our dataset the same files that Linguist discard when computing language statistics, e.g., documentation and vendored (or third-party) files. As a result, we automatically removed 129,455 files (34%), including 5,125 `.js` files, 3,099 `.php`

[2]https://github.com/github/linguist

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015

files and 2,049 .c files. After this automatic clean up step, we manually inspected the first two top-level directories in each repository, mainly to detect third-party libraries and documentation files not considered by the Linguist tool. As a result, we manually removed 10,450 files.

Handling Aliases: A second challenge when inferring code authorship from software repositories is to detect alias (i.e., different IDs, for the same developer). To tackle this challenge, we first consider as coming from the same developer the commits identified with different developers' names, but having the same e-mail address. Second, we compared the names of the developers in each commit using Levenshtein distance [5]. Basically, this distance counts the minimum number of single-character edits (insertions, deletions or replacements) required to change one string into the other. We considered as possible aliases the commits whose developers' names are distinguished by a Levenshtein distance of just one. We then manually checked these cases, to confirm whether they denote the same developer or not.

2.3 Truck Factor

To calculate the Truck Factor, we use a greedy heuristic: we consecutively remove the author with more authored files in a system, until more than 50% of the system's files are orphans (i.e., without author). Therefore, we are considering that a system is in trouble if more than 50% of its files are orphans.

3 Results

Table 2 presents the Truck Factor (TF) we calculated for the analyzed GitHub repositories. The results in this table are summarized as follows:

- Most systems have a small truck factor:
  - 61 systems have TF=1 (46%), including systems such as MBOSTOCK/D3, and CLOJURE/CLOJURE.
  - 37 systems have TF=2 (28%), including systems such as CUCUMBER/CUCUMBER, MRDOOB/THREE.JS, MOZILLA/PDF.JS, SPRING-PROJECTS/SPRING-FRAMEWORK.
- The two systems with the highest Truck Factor are TORVALDS/LINUX (TF = 90) and HOMEBREW/HOMEBREW (TF = 159). Homebrew is a package manager for the Mac OS operating system. The system can be extended by implementing formulas, which

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015

Table 2: Truck Factor results

TF	Repositories
1	activeadmin/activeadmin, alexreisner/geocoder, atom/atom-shell, bjorn/tiled, bumptech/glide, caskroom/homebrew-cask, celery/celery, celluloid/celluloid, clojure/clojure, codemirror/codemirror, dropwizard/dropwizard, dropwizard/metrics, elasticsearch/logstash, erikhuda/thor, eugeny/ajenti, getsentry/sentry, github/android, gruntjs/grunt, jadejs/jade, janl/mustache.js, jnicklas/capybara, jrburke/requirejs, justinfrench/formtastic, kivy/kivy, koush/ion, kriswallsmith/assetic, leaflet/leaflet, less/less.js, mailpile/mailpile, mbostock/d3, meskyanichi/backup, mitchellh/vagrant, mitsuhiko/flask, mongoid/mongoid, nate-parrott/flashlight, netty/netty, nicolasgramlich/andengine, omab/django-social-auth, openframeworks/openframeworks, paulasmuth/fnordmetric, phacility/phabricator, plataformatec/devise powerline/powerline, puphpet/puphpet, pydata/pandas, ratchetphp/ratchet, reactivex/rxjava, sampsyo/beets, sandstorm-io/capnproto, sass/sass, sebastianbergmann/phpunit, sferik/twitter, silexphp/silex, sparklemotion/nokogiri, sstephenson/sprockets, strongloop/express, substack/node-browserify, thinkuplc/thinkup, thoughtbot/factory_girl, thoughtbot/paperclip, wp-cli/wp-cli
2	ajaxorg/ace, ansible/ansible, apache/cassandra, bbatsov/rubocop, bundler/bundler bup/bup, composer/composer, cucumber/cucumber, divio/django-cms driftyco/ionic, drupal/drupal, elasticsearch/elasticsearch, excilys/androidannotations, facebook/osquery, facebook/presto, friendsofphp/php-cs-fixer, github/linguist, haml/haml, itseez/opencv, jashkenas/backbone, jekyll/jekyll, johnlangford/vowpal_wabbit, jquery/jquery-ui libgdx/libgdx, moment/moment, mozilla/pdf.js, mrdoob/three.js, prawnpdf/prawn respect/validation, rg3/youtube-dl, sfttech/openage, spring-projects/spring-framework, thinkaurelius/titan, thumbor/thumbor wordpress/wordpress, xetorthio/jedis, yiisoft/yii2
3	bitcoin/bitcoin, boto/boto, bvlc/caffe, gradle/gradle, ipython/ipython jquery/jquery, meteor/meteor, shopify/active_merchant, spotify/luigi
4	chef/chef, cocos2d/cocos2d-x, emberjs/ember.js, iojs/io.js, ruby/ruby
5	diaspora/diaspora, django/django, joomla/joomla-cms, resque/resque, tryghost/ghost
6	puppetlabs/puppet, scikit-learn/scikit-learn
7	rails/rails
8	git/git , jetbrains/intellij-community, seldaek/monolog, v8/v8, webscalesql/webscalesql-5.6
9	saltstack/salt
10	fog/fog
11	odoo/odoo, php/php-src
12	android/platform_frameworks_base
21	fzaninotto/faker
90	torvalds/linux
159	homebrew/homebrew

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015

are recipes for installing specific software packages. Homebrew currently supports thousands of formulas, which are typically implemented by the package's developers or users, and rarely by Homebrew's core developers. For this reason, the system has one of the largest base of contributors on GitHub (almost 5K contributors, on July, 14th, 2015). All these facts contribute for Homebrew having the largest Truck Factor in our study. However, if we do not consider the files in `Library/Formula`, HomeBrew's Truck Factor decreases to just one.

Acknowledgment

Our research is supported by CNPq and FAPEMIG.

References

[1] L. Williams and R. Kessler, *Pair Programming Illuminated*. Addison-Wesley, 2003.

[2] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: Modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, pp. 14:1–14:42, 2014.

[3] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *32nd International Conference on Software Engineering (ICSE)*, 2010, pp. 385–394.

[4] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *22nd International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 155–165.

[5] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.

PeerJ PrePrints | https://dx.doi.org/10.7287/peerj.preprints.1233v1 | CC-BY 4.0 Open Access | rec: 14 Jul 2015, publ: 14 Jul 2015