

Quo Vadis, Open Source? The Limits of Open Source Growth

MICHAEL DORNER, Blekinge Institute of Technology, Sweden

MAXIMILIAN CAPRARO, Friedrich-Alexander-University Erlangen, Germany

ANN BARCOMB, University of Calgary, Canada and Friedrich-Alexander-University Erlangen, Germany

Open source software plays a significant role in the software industry. Prior work described open source to be growing polynomially or even exponentially. However, such growth cannot be sustained infinitely given finite resources. In this study, we present the results of four accumulated measurements on size and growth of open source considering over 224,000 open source projects for the last 25 years pulled from Open Hub. For each of those projects, we measured lines of code, commits, contributors and lifecycle state over time, which reproduces and replicates the measurements of three well-cited studies. In our sample, we found the number of active open source projects has been shrinking since 2016 and the number of contributors and commits has decreased from a peak in 2013. We discuss multiple rivaling interpretations for our observations. Because the results were unexpected to us, we encourage further research using alternative data sets.

CCS Concepts: • **Software and its engineering** → **Open source model**; **Software evolution**; • **Social and professional topics** → *Sustainability*.

Additional Key Words and Phrases: open source, growth, limits, replication study, evolution

1 INTRODUCTION

Open source software is everywhere in modern software development: Open source development tools help to build software and open source components are used as part of other software. Open source is recognized to be capable of delivering high quality software [10]. This makes open source components an integral part of modern software industry.

Most studies on the growth of open source projects are vertical and have investigated the growth of single, mostly extraordinary, successful, or special open source projects [19, 37, 41, 49], or, on occasion, software forges [46].

Three horizontal studies investigated the growth of open source as a whole: [7] from 2003, [26] from 2007, and [12]. From those three prior studies, open source is expected to grow quadratically [26] or exponentially [12] with respect to lines of code and number of projects. But obviously, no system with a polynomial or even exponential growth rate can remain stable: At some point it might plateau or reach saturation, because all available resources are exhausted.

This study replicates the measurements of project-specific quantities suggested by the three prior studies (lines of code, lifecycle state), but also reproduce the measurements by new measurands (contributors, commits) on an enlarged and updated data set of 180 million commits contributed to more than 224,000 open source projects in the last 25 years. In this way, we evaluate existing growth models and help to mature the knowledge of open source by addressing both internal and external validity.

In detail, the contributions of this paper are

- the definition and discussion of four measurements and measurands for quantifying open source including a novel open-source project life-cycle model,

Authors' addresses: Michael Dörner, michael.dorner@bth.se, Blekinge Institute of Technology, Karlskrona, Sweden, 37179; Maximilian Capraro, maximilian.capraro@fau.de, Friedrich-Alexander-University Erlangen, Erlangen, Germany, 91054; Ann Barcomb, ann@barcomb.org, University of Calgary, Schulich School of Engineering, Calgary, Canada, T2N 1N4, Friedrich-Alexander-University Erlangen, Erlangen, Germany, 91054.

Table 1. Three prior studies on the evolution of open source, showing data source, project sample size, and date range.

	Study	Year	Source	Project sample size	Time frame
[7]	Study A	2003	FreshMeat.net	406	? – 2002-07-01
[26]	Study B	2007	SourceForge.net	4,047	unknown
[12]	Study C	2008	Ohloh.net	5,122	1995-01-01 – 2006-12-31

Table 2. All hypotheses extracted from studies A, B, and C, along with the types of measurements from this study.

	Hypothesis	Study	Measurement
1	Open source grows with respect to byte size.	A	byte size
2	Open source grows quadratically with respect to lines of code.	B	lines of code
3	Open source grows exponentially with respect to lines of code.	C	lines of code
4	Open source grows exponentially with respect to projects.	C	projects

- the multi-dimensional measurements using Open Hub as a measuring system to measure the growth of open source with respect to lines of code, commits, number of contributors, and stateful projects, and, thereby,
- the replication and reproduction of the measurements by three prior studies.

Throughout this paper, we are following the terminology of metrology [23]. Therefore, we differentiate between replication (different team and same experimental setup) and reproduction (different team and different experimental setup). In this, we follow the definition of ACM, which is based on [23], and remove the ambiguity of replication and reproduction [34], too.

Three studies investigated the evolution and growth of open source as a whole [7, 12, 26]. All three studies are sample studies aiming for generalizability over the population of open source projects which means the focus is not on specific contextual details. This implies a sample study research strategy [48]. This also means that all results must also hold true for a larger or newer sample of open source.

The following structure guides the reader through the study: In Section 2, we present all three studies and extract their measurement results as hypotheses. In Section 3, we describe our measurements, which not only include measurands from the prior studies, but also incorporate an additional measurand: contributors. Section 4 describes our approach to ensuring the correctness of the data. In Sections 5 and 6, we present and discuss the results of our measurements, while in Section 7 we consider possible interpretations of our findings and discuss future work. In the last section, we summarize and conclude our findings.

2 STATE OF THE ART

In this study, we replicate the measurements of three studies: [7] from 2003 (Study A), [26] from 2007 (Study B), and [12] from 2008 (Study C). Table 1 lists all three studies with their time of publishing, their source, the number of open source projects in the sample, and the considered time frames.

From those three prior works, we extracted four measurements and hypotheses, which are either explicit or implicit hypotheses, which are listed in Table 2.

The original data sets of all studies were not publicly archived by the authors and, in case of Study A and B, the data sources no longer exist. We contacted all lead authors from prior studies to clarify inconsistencies or ambiguities in their work, but only one author team responded.

2.1 Study A

Study A [7] from 2003 presented a descriptive longitudinal study on 406 open source projects and analyzed 12 different quantities of open source projects. Although measured three times (February 2001, January 1st, 2002, and July 1st, 2002), all quantities are statically analyzed containing the whole time frame only and are not presented continuously over time or at least over those three or four sampling points. The exact sampling point (1999) and the date of first data collection (also 1999 or February 2001) remains unclear.

A pseudo-sampling was applied: the study uses the project status defined by the FreshMeat platform, an index for open source software, with the status *planning*, *pre-alpha*, *alpha*, *beta*, *stable*, *mature*. It is unclear how this status is determined. For each possible status, half of the projects were randomly selected, resulting in 406 projects, according to the authors half of the population in 1999 hosted on FreshMeat. However, the sample could be drawn from “living” or inactive open source projects, or both. Neither living nor inactive is defined.

The quantities number of developers, size, modularity level, documentation level, and success of a project are used, but not properly defined and described in the study. All derived quantities computed for the study lack a precise definition, i.e. success of project and number of developers.

Study A measured the byte size of the source code excluding “documentation and unessential binary or code files, such as HTML, GIF, JPG”. The study found that 63% of the examined projects have not changed their size over six months, with 34% of projects changing less than 1%. In conclusion, only 3% of all projects changed more than 1% in size over six months. Over the longer period of time encompassed by the first and last samplings, 59% of projects did not change size, 22% grew by up to 10%, 15% grew between 10%-50% in size, and 5% more than doubled in size. The authors observed that open source grows:

Hypothesis 1

Open source grows with respect to size (in bytes).

They neither discussed the magnitude of growth nor made predictions about future growth.

The study considered additional measurements. We were not able to replicate or reproduce these measurements because no detailed description of their construction was given: The original data source FreshMeat does not exist anymore: On October 29, 2011, FreshMeat was renamed to Freecode. Since June 18, 2014, Freecode is no longer maintained. The quantities age, application domain, programming language, size, number of developers, number of users, modularity level, documentation level, popularity, and vitality rely on computations by the portal owner or are derived from those. Their construction is unclear and, in consequence, we cannot replicate or reproduce the measurements.

The measurements for vitality and popularity are explicitly discussed by the study. According to this study, 23% of projects increased their vitality. Vitality V is defined by

$$V = \frac{R \cdot A}{L} \quad (1)$$

where – according to description – R is the number of releases in a given period (t), A is the age of the project in days, and L is the number of releases in the period t . We assume a typo in the formula and/or the definitions: From the definition, $R = L$ and, therefore, $V = A$ in equation 1. We also miss the portion of constant vitality in order to estimate the growth.

90% of projects did not change their status in a period of six months. How the status is computed/defined and what are the six months remains unclear.

60 projects out of 400 (15%) are active projects, the rest are considered lethargic. Study A defines an active project to have an increasing “vitality, popularity and subscribers and developers”. All projects which are not active are defined as lethargic. It remains unclear if all three measurands must be increasing (\wedge) or at least one of them (\vee) for a project to be deemed active.

Popularity P is defined by

$$P = \sqrt[3]{\frac{U \cdot R \cdot (S + 1)}{3}} \quad (2)$$

where U stands for the count of visit to the project homepage, R is the number of visits to the project on FreshMeat pages, and S is the number of subscribers. It is unclear to what extent the study controlled for bots and web crawlers, and what measures were taken to merge duplicate user identities. According to the description, P is normalized between 0% and 100%, which obviously is not true for any $U, R, S \geq 1$, unless an unspecified adjustment was subsequently applied.

2.2 Study B

Study B [26] from 2007 found the growth of open source to be best described as a quadratic function for a sample of 7,734,082 commits with 663,801,121 lines of code added and 87,405,383 removed from 8,621 projects contributed by 12,395 developers. The study did not discuss whether the lines of code count includes comments or not. Only CVS projects are considered. Neither the study B nor the supplementary work [25] contains any information on the time when the sampling took place and the considered time frame.

In detail, the authors found the growth better described as a quadratic

$$S_B(t) = a \cdot t^2 + t \cdot b + c$$

than as a linear function

$$S_A(t) = a \cdot t + b$$

where S_B and S_A , respectively, is the size in lines of code at time t as days after the first commit. Both models are evaluated by the adjusted R^2 value, which is not applicable for non-linear regressions like this [45]. Study B observed:

Hypothesis 2

Open source grows quadratically with respect to lines of code.

2.3 Study C

Study C [12] from 2008 found an exponential growth of open source with respect to source lines of code, and number of new and total number of open source projects. Comment lines and empty lines are excluded from the lines of code count.

Study C considers the 5,122 most popular open source projects according to the number of in-links provided by the Yahoo! search engine to their website. A list of those open source projects is not available to us. In contrast to study A and B, the original data source, Ohloh (now Open Hub) is still available.

Study C excluded “all commits where lines of code added is greater than average code added per commit plus three times the standard deviation”. The exact measurement is not completely described in the paper: Although labeled in the plots, the description of the approach indicates that not lines added, but the net change with respect to lines of code is considered. The measurement of total number of projects is redundant to the measurement of newly added projects, because for a population growing at exponential rate, the removal rate from the population must be smaller. This is a sufficient and necessary condition for an exponential growth. However, if one considers lines of code added only, the lines of code added before the sampling period are removed and older projects are disadvantaged.

Study C used R^2 to evaluate the goodness of the models, which is not an adequate measure for the goodness of fit in non-linear models [45]. Additionally, the curve fitting and its initial parameters are not presented.

We conclude the following two hypotheses, which are derived from the findings of study C:

Hypothesis 3

Open source grows exponentially with respect to source lines of code.

Hypothesis 4

Open source grows exponentially with respect to number of projects.

3 MEASUREMENTS

As our primary measurement system we rely on Open Hub¹, which was well-known under the name Ohloh until August 2014. Open Hub is an online platform which provides an infrastructure to crawl and index open source projects from different sources. Open Hub’s crawlers support all main open source version control systems: git, CSV, SVN, Bazaar, and Mercurial. This allows it to collect commit information across different source code hosts like GitHub, Gitlab, BitBucket, SourceForge, etc. The data source also includes deleted open source projects and their commit history, which allows a retrospective view on the development of open source.

In addition to study C, [32] used Ohloh develop a classification to assess the quality of a sample of open source projects and to identify open source projects that could be added to improve the sample quality. Their sample consisted of 20,028 active open source projects, where active is defined as at least one commit per year.

Open Hub provides information on each project of the cumulative sum per month of

- total and added/removed code lines,
- total and added/removed comment lines,
- total and added/removed blank lines,
- commits, and
- contributors.

¹We are not referring to the data collection tool with the same name described in [13].

Table 3. All measurements extracted from Study A, B, and C, along with the types of measurements from this study.

Prior measurement		Lines of code	Commits	State	Contributors
Size	(Study A)	reproduction			reproduction
Lines of code	(Study B, C)	replication	reproduction		reproduction
Projects	(Study C)			replication	reproduction

We crawled those data points through a REST API with XML responses from Open Hub for each project in the sample. Additionally, we gathered the project’s Open Hub page as HTML in order to parse all duplicate warnings, because this information is not available through the REST API. Our crawling and analysis tool chain as well as the anonymized data set is fully published on Zenodo and GitHub.

We evaluate hypotheses 1, 2, and 3 by measuring lines of code, and hypothesis 4 by measuring stateful projects: We replicated measurements of Study B and C on lines of code from, and measurement of Study C on stateful projects. We also reproduce measurements of Study A on the size (measured in bytes) from Study A through measuring lines of code and measurements of Study B and C on lines of code through measuring commits. As we are measuring human activities, we also reproduce all measurements by measuring contributors. Table 3 lists all hypotheses we are evaluating, the measurements we conducted in this study, and whether we replicate or reproduce the prior measurements.

3.1 Lines of Code

A *line of code* (LoC) is a non-blank line containing either comments, source code, or both. A *source line of code* (SLoC) is a non-blank line starting with source code. A *comment line of code* (CLoC) is a non-blank line containing comments only.²

Study B and C estimated the size of available source code in open source by measuring lines of code. Study C excluded comment lines of code. It remains unclear if Study B included or excluded comment lines. However, a comment is a valuable contribution to a software project. In modern programming languages like Go or Python, comments are directly embedded in the source code for documentation purposes. Therefore, we replicate the measurements of both Study B and C by measuring lines of code, which contains both comment and source lines of code. However, we also distinguish between source and comment lines of code during our measurements.

Study A measured the byte size of the source code excluding “documentation and unessential code, such as HTML, GIF, JPG”. Because the exact inclusion and exclusion criteria and definition of unessential code are not available or described by Study A, we are reproducing this measurement by also measuring lines of code: Any non-empty line of code has at least one byte. Therefore, our measurement is consistent with Hypothesis 1, which states that open source grows (in bytes).

By measuring lines of code and its derivatives source lines of code and (indirectly) comment lines of code, we are able to replicate the measurements of all three studies.

Open Hub evaluates the type of lines of code using *ohcount*, which is publicly available³. This tool can only distinguish binarily between code and comment (and blank) line. However, a line with both source code and followed by a comment is classified as lines of source code – although having both.

²This definition also applies to inline comments such as `/* some comment*/ int i = 0;` in C/C++, also at the beginning.

³<https://github.com/blackducksoftware/ohcount>

A more fine-grain split in lines of code added, removed, or changed is not universally applicable. This derivative of LoC has its origin in *diff*, a Unix command line tool to calculate and display the line-based difference between two files. Using *diff*, a line can be added, removed, or changed. However, those status changes are not easily detectable: due to optimizations, two files with three lines each a, b, c and c, b, a, can lead to three different outputs – depending which character is detected as common and this depends on the version, the system, and optimization in use. Thus, we do not rely on any diff-generated information and do not follow the experimental setup of Study C strictly. Instead, we use the monthly measurements on line of code on the project.

3.2 Commits

Study C assumes “a positive correlation between work spent on open source, its total growth in terms of code and number of projects, and the revenue generated from it, understanding the overall growth of open source will give us a better indication of how significant a role open source will play in the future.” However, the choice of programming language, tooling, coding style guidelines, and embedded documentation has a large impact on lines of code. For example, a GitHub client component written in Python has 1,113 lines of code in 12 .py files, a comparable library in Go has 60,438 LoC in 207 .go files. However, we believe that measuring commits are more suitable for estimating effort spent in open source in total.

A *commit* is a semantically enclosed and authored tailored contribution to a software project. Commit is the predominant term across most version control systems such as git, CVS, SVN, Mercurial, and Bazaar. Synonyms are, for example, *change set* in Microsoft’s Team Foundation Server (TFS). Unlike commits in data management (e.g. databases), commits in version control systems are persistent and kept in the repository. Each commit contains, in addition to other information, meta information on

- the code change to be contributed,
- a unique identifier,
- the timestamp of the commit action, and
- the author of the commit.

Several studies used commits for effort estimation [6, 50] or to measure collaboration among organizational boundaries [9]. Using commits as the basis for estimating effort is supported by research showing that the interval between consecutive commits does not vary widely [28, 31], implying that developers typically put about the same amount of time into each commit. When commit size is calculated as the total number of commits in a time period, it follows a power-law distribution [30].

Using the OpenStack project as a case study, [38] proposed an approach to estimating development effort based on identifying full-time paid contributors and calculating the number of person-months they contributed in aggregate, assuming 40-hour work weeks [38]. The total time was estimated by extrapolating the output of full-time contributors to cover the remaining proportion of contributions.

Open source software development has become, in a large part, commercial paid-for software development [36]. About 50% of all commits to projects to our sample were authored during regular (Western) working hours, Monday to Friday, 9 am to 5 pm (local time) [36]. This implies that commits are not only related to labor, but also to money. Attributing a value to effort is beyond the scope of the current study, however.

Commit size calculated in lines of code has also been described as following a power-law distribution [2] or a Pareto distribution [21, 27]. The majority of commits affect 2-4 files, 6-46 lines of code, or 2-8 sections of contiguous lines of code [1].

Therefore, we use the measurement of commits for reproducing measurements of lines of code.

3.3 Lifecycle State

Study C measured the number of projects and new crawled projects. However, purely counting the number of open source projects falls short: In contrast to classical software projects, open source projects do not have a pre-defined scope, thus an explicit beginning and end. Although an open source project may not be actively developed any more and abandoned, the code and all related artefacts such as documentation and communication may be still publicly available. In order to differentiate between active, inactive, abandoned, and deleted open source projects, a life-cycle model must be considered – traditional, closed-source life-cycle models do not fit to open source projects [42]. Therefore, we present in this section a novel open source project life-cycle based on a project’s commit frequency.

3.3.1 Related work. Before we define our novel open source project life-cycle model, we would like to give an overview of the existing open source life-cycle models and discusses their non-applicability.

[8] presented an empirical study on two OSS projects, *Arla* and *Wine*, to illustrate different phases in their lifecycle [8]. The basic measure for the determination of the phases of project life-cycle, cathedral and bazaar phase, and the transition thereof, is number of unique developers and number of produced modules or subsystems. Once an open source project transits to the bazaar phase, the way back is not given. This might apply to the two selected open source projects, but does not generalize: Open source can be actively developed or abandoned within the bazaar phase, or may fail to reach the bazaar phase [33, 44, 52].

The open source life-cycle project by [43] lists three major stages: project initiation, going open, and project growth or decline [43]. It “is based on is based loosely on the discussion on the OS development lifecycle provided by [14]”. 14 non-programming and 15 programming open source project were selected, but only the latter are considered. Although not naming it explicitly, we assume an adopted case study research method. A mentioned *case coding form* (assumingly a case study protocol), which was developed for this study and should guide the analysis, is not provided. None of criteria or metrics are available or defined.

[53] proposes a qualitative model of open source project life-cycle containing the stages introduction, growth, maturity, decline, or revive. All stages are qualitatively described, but not defined. The underlying measure is number of downloads. However, number of downloads does not imply the use of an open source project and can be hardly measured.

[29] suggests a maturity life-cycle model based on development activity with the stages born, childhood, adolescence, adulthood, and obsolete and applied it to an unknown number of open source projects of two major OSS communities, Apache Software Foundation and the OpenStack Foundation. None of the states are defined or described in detail.

Not a complete life-cycle model, but a definition of an active open source project is provided by [28], which is based on [11]: “A project is active at a given point in time if the number of commits in the preceding 12 months is at least 60% of the number of commits in the 12 months before that.” However, the definition applies only for projects older than 25 months, and relies on numbers of [11] which are arbitrary. This makes this definition not applicable for this study.

In their descriptive longitudinal study, [7] used a data source containing *living* and *inactive* open source projects [7]. Neither the state active/living nor inactive is defined or described.

Open source foundations like the Eclipse [16] or Apache Software [15] foundation typically explicitly define life-cycle models with the goal to describe how their open source projects mature through the ranks (e.g. from a proposed project to a mature project). By doing so, they implicitly also model whether a project is active or abandoned. The model we propose explicitly describes the activity of a project and does exclude project maturity.

3.3.2 Definition. In this subsection, we present a novel open source life-cycle model, where the transition of the states are determined by the commit frequency for a given time frame.

The commit frequency denoted by $c_p(t)$ of a project $p \in P(t)$ at timeframe t is the number of occurrences of commit events per timeframe t . $P(t)$ is the set of available open source projects at timeframe t .

Our **open source life-cycle model** is a non-deterministic finite automaton (NFA) and is formally represented by the 5-tuple $(Q, \Sigma, \delta, \{\text{active}\}, F)$, consisting of

- the finite set of states

$$Q = \{\text{active}, \text{inactive}, \text{abandoned}, \text{deleted}\}$$

- the finite set of input symbols as alphabet

$$\Sigma = \{\oplus, \ominus, \oslash, \varepsilon\}$$

computed the commit frequency $c_p(t)$ at timeframe t where

$$\oplus := c_p(t) \neq 0,$$

$$\ominus := c_p(t) = 0, \text{ and}$$

$$\oslash := p \notin P(t),$$

- a transition function

$$\delta : Q \times \Sigma \rightarrow Q,$$

- the start state $\{\text{active}\}$, and
- the set of final states

$$F = \{\text{abandoned}, \text{deleted}\}.$$

Figure 1 is a graphical representation of the NFA-based open source project life-cycle.

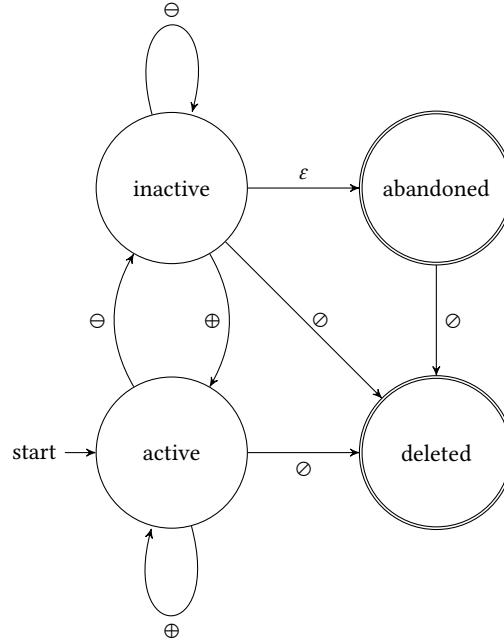
A project starts with an initial commit. If it receives contributions (\oplus) it remains active, if not (\ominus) it transits to the state inactive. It is common that – even after a long period of inactivity – a project can be reactivated with a contribution. However, some projects are just dumped or archived and not actively developed any more; they are in the state abandoned. The transition (ε) is not explicit, but happens usually after a long time of inactivity. An open source project can be removed or deleted from the public repositories (\oslash) at any point in time. Those projects are not accessible any more and are vanished from the open source population. For this study, we set the time frame for any commit – independent of its human origin – in the life-cycle to be one calendar month. This time period is arbitrary.

This measurement aims to replicate the measurement on projects by Study C.

3.4 Contributors

Because all of the previous measurements are human-based activities (writing code, starting a new open source projects, or contributing a commit), we would also like to extend the existing measurements by measuring contributors and, by that, reproduce all other measurements.

Fig. 1. The open source project life-cycle. The project starts with an initial commit. The model is independent of the time frame chosen for commits.



A *contributor* is a role of a person who contributes to a project. In this paper, we focus exclusively on code contributions and accepted changes to the project’s source code. In open source, usually only committer can accept changes, because only they have write-access to the repository. A human being can serve as multiple contributors, for example, depending on their professional affiliation. Open Hub is able to map contributor identities to a person. However, this mapping requires a registered user and is in general not trivial.

4 FILTERING

At the time of the measurement (from 2020-01-02 to 2020-01-04), Open Hub lists 477,851 open source projects. The development activity is available for 224,884 projects only.

4.1 Timeframe

We limited the analysis time frame from 1994-01-01 to 2018-12-31. Incorrect time configuration on developers’ machine can cause a wrong timestamp for the commit, either accidentally (e.g., Unix time, which is the Thursday, 1 January 1970) or on purpose⁴. Also Open Hub assigns 1970-01-01⁵ if there is no valid timestamp. The default Unix timestamp (1970-01-01) is already excluded from the sample anyway. We bypass this issue by limiting out analysis from the time

⁴For example, according to its Git-history, the Go programming language started back in 1972 with a commit (commit hash 7d7c6a97f815e9279d08cfaea7d5efb5e90695a8) by Brian Kernighan with a C-code snippet – obviously a remembrance for Kernighan’s famous *hello world* memo.

⁵https://github.com/blackducksoftware/ohloh_scm/blob/ca0e512177fb9958473812445d6a54b551b3ce9b/lib/ohloh_scm/git/activity.rb#L215

frame from 1994-01-01 to 2018-12-31. Those exceptions aside, we assume the timestamps to be correct by default. Smaller deviations (e.g., by setting up the time manually, wrong time zones, etc.) are intercepted by the monthly resampling.

4.2 Duplicates

Some projects contain commits which originate from other projects. We extracted the information on duplications from all the collected projects' HTML pages as it is not accessible through the REST API and excluded the projects marked as duplicates. We relied on the duplication detection by Open Hub, which is not publicly available.

We excluded 731 duplicates from 484 original projects. The Linux Kernel was the most duplicated project with 79 duplicates.

4.3 Outlier detection

In our data set, we found outliers. For example, we found values up to 453,380 commits by one contributor in November 2018 within the *Beagle Board* project⁶: A large-scale change to a database and to log files was split into tiny commits each with some dozen added lines on one specific file.

Statistical outlier detection such as z-score, or median absolute deviation (MAD) fails in detecting outliers for our data set, because most stellar, large open source projects such as OpenStack, KDE, Linux kernel, Chromium, or Firefox are considered as outliers with respect to contributors, commits, and lines of code, but are regular, and very successful open source projects.

Therefore, we decided not to apply any statistical outlier detection and exclude the one occurrence at the BeagleBoard project. However, we present the median for the relevant measurements (lines of code, commits, and contributors) to mitigate the outliers problem. Additionally, we also present the mean, because the difference between the median and the mean is useful to characterize how skewed the data is.

5 RESULTS

In this section, we present the results of our analysis on the accumulated measurements of lines of code, commits, lifecycle state, and contributors over all 224,844 open source projects in the sample. For describing time frames, we define *until* and *since* to be inclusive.

Lines of code and lifecycle result in a physical artefact, code and projects. Thus, they are presented as cumulative sums as they do not disappear, but remain reusable. Commits and contributors are not reusable, they describe work effort.

In all four measurements, we found an initial, transient exponential growth. This growth exponential growth is evaluated by using an exponential function $y = a \cdot \exp(b \cdot x) + c$ with the initial parameter $a, b, c = 0$ with a resolution of full years. Those exponential models are depicted in all relevant graphs by a dashed red line. The exact parameters are published in the related Jupyter notebooks.

5.1 Lines of code

The body of open source lines of code is tremendous: more than 17 billion lines of open source code are available today. As of the end of 2018, open source projects contained 17,586,490,655 lines of code, comprised of 14,588,351,457 lines of source code and 2,998,139,198 lines of comments.

⁶<https://github.com/beagleboard/beagleboard-org>

Figure 2 depicts an aggregated view on lines of source code and comment lines of code over time, with the exponential model for lines of code.

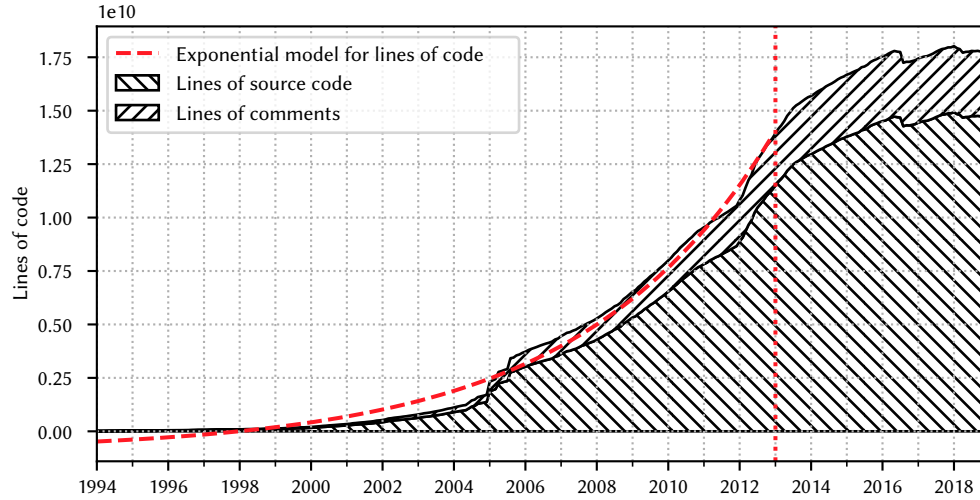


Fig. 2. Lines of code over time, showing lines of code and combined lines of comments and code. An exponential curve no longer fits the combined comments and code after 2013, while the source code growth rate declines after 2011.

Observation 1

Although initially growing exponentially through 2010, the growth in lines of code has continuously slowed down since 2011.

↗ Figure 2

Although neither quadratic nor exponential, open source still grows with respect to lines of code. However, since 2011, the growth has continuously slowed down. The slow-down is greater for lines of code containing source code than comments: The portion of comment lines increases over time. We speculate that the relative increase of comment lines correlates with the popularity of programming languages and tools which generate documentation from source code (e.g. JavaDoc for Java, Pydoc for Python, or Godoc for Go).

Figure 3 shows that the total amount of lines of code in open source has been stable since 2011. The mean monthly size in lines of code per project is stable around 83,000 lines of code. The median of lines of code decreased over the same period of time from 2700 to 2200 lines of code. This implies that the outlier, stellar project can compensate the decrease for open source as a total – the growth is not distributed among all project sizes.

5.2 Commits

In the last 25 years, the 224,342 open source projects received 180,937,525 commits in total. Figure 4 shows the total number of commits over time.

Again, we found an exponential growth of commits until 2010 and can confirm the measurement results of lines of code by reproduction.

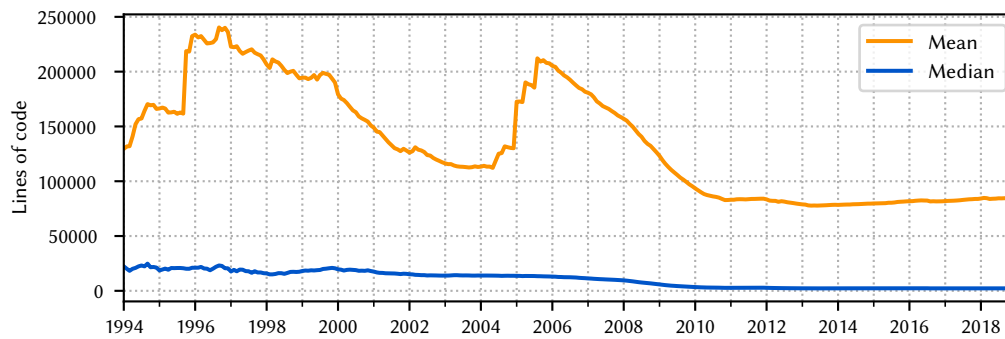


Fig. 3. Mean and median of lines of code.

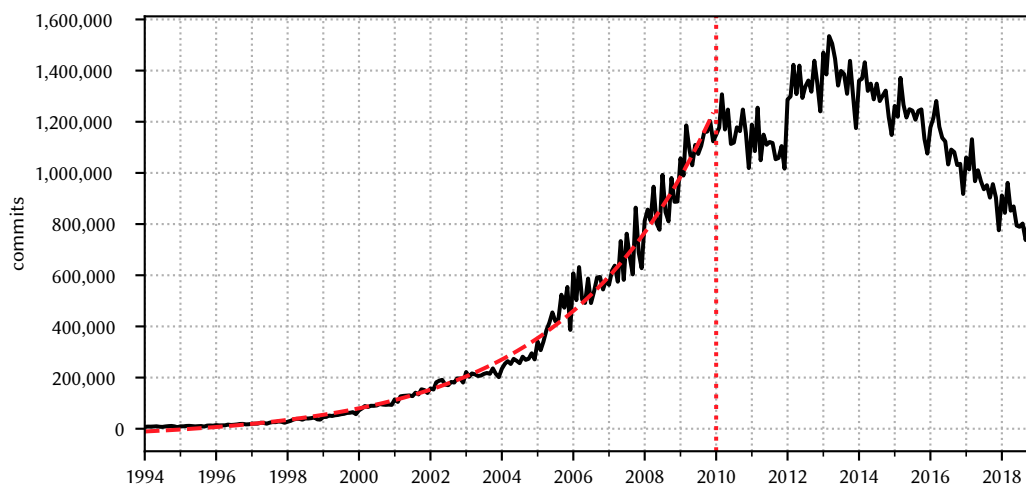


Fig. 4. Number of new commits made over time. The exponential growth curve does not fit after 2010.

Observation 2

Although the number of open source commits grew exponentially until March 2010 and peaked in 1.5 mio commits in March 2013, it has since declined. In 2018, it reached a level comparable to the end of 2007.

↗ Figure 4

After a dent in 2010 and 2011, the commits monthly contributed to open source reached its peak with 1,534,726 million commits in March 2013. With seasonal minima in December from 2013 to 2019, the trend of commits contributed to open source is downwards.

Observation 3

The mean and median of monthly commits per projects is decreasing.

↗ Figure 5

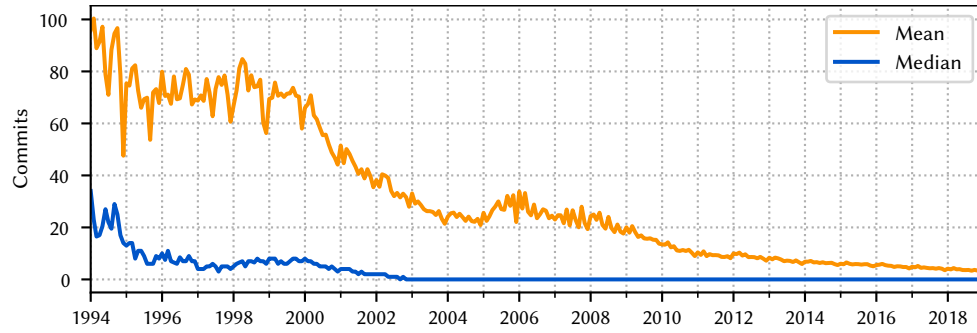


Fig. 5. Mean and median of commits per project over time. The mean number of commits has declined fairly consistently since 2000, while the median stabilized at zero in 2003.

As depicted in Figure 5, the number of commits for each project is decreasing. The median of monthly commits per open source project reached zero in 2003 and has not increased since then. This means more than half of the known projects have not received a monthly contribution since 2003. The mean of monthly commits per open source project continuously decreases: After stable phases between 1994 and 2000 with ca. 70 commits in average per project and between 2005 and a local maximum around 2006, the mean as of end 2018 is 3.05 commits per project per month.

5.3 Lifecycle State

We evaluated the projects using the lifecycle model presented in Section 3.3. We used one month as the time period for evaluating the underlying life-cycle. As of the last month of measurement (December 2018), we found 224,342 open source projects. 196,009 of them were inactive, 13,085 where abandoned, and 58 new projects were added. 15,046 were deleted over the whole time frame of 25 years. Figure 6 depicts the lifecycle states over time.

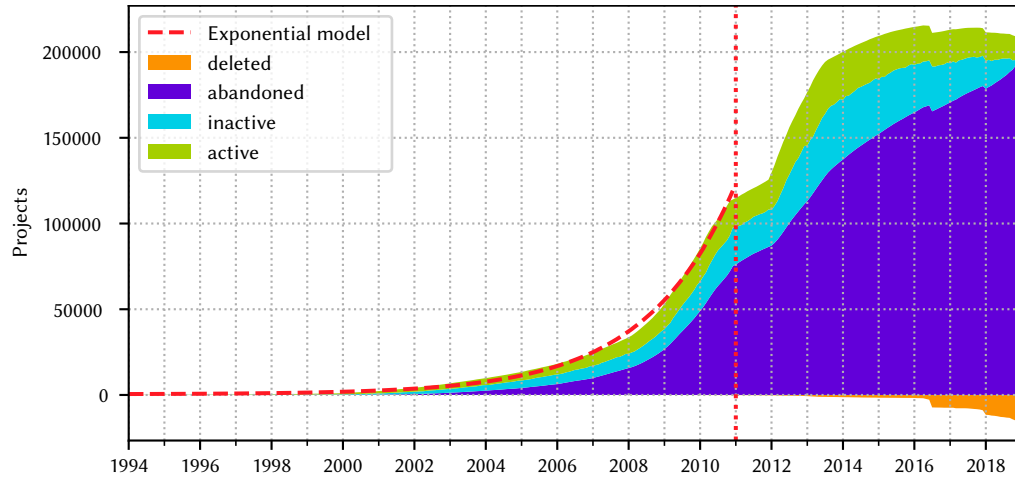


Fig. 6. Available open source projects by lifecycle phase, where the time for implicit state changes is one month. While the exponential curve fits until 2011, this is only when all projects, including inactive and abandoned ones, are included. The number of abandoned projects relative to active projects continues to grow.

Again, we were able to confirm an exponential growth until 2013 over all available projects, while most of the projects are inactive – they do not receiving a commit within a given month.

Observation 4

Until July 2013, the available open source projects grew exponentially, reaching a plateau in 2016.

↗ Figure 6

We observed that most open source projects are inactive. Beyond that, most inactive projects never receive a contribution again – they are abandoned. The growth of abandoned projects in the last year comes from the end of measurement in 2018: Because the measurement ends in 2018, all inactive projects are abandoned because they have not received a contribution since then.

Observation 5

The vast majority of open source projects are abandoned; they have never received a contribution again until the end of measurement or their deletion.

↗ Figure 6

The vast majority and a constant portion of open source project is abandoned. The portion of actively developed open source projects which receive at least one contribution in a given month is small and approximately constant over time.

Observation 6

We observe a significant decrease of new projects in the year 2011.

↗ Figure 6

There is a significant drop in new projects with their first commit in 2011, which we are not able to explain. Without this drop, open source could probably have retained an exponential growth in projects until 2012.

5.4 Contributors

Again, we observed an exponential growth of open source contributors until 2010. Figure 7 illustrates our observations.

Like for the monthly commits, the number of monthly contributors decreased after the peak in 2013. We also can see a yearly, seasonal drop in December around Christmas.

Observation 7

Although growing exponentially until March 2010 and reaching its peak in March 2013 with 107,915 contributors, the number of open source contributors has, as of 2018, decreased to the level of 2008.

↗ Figure 7

We refer the reader to Figure 7 and the sharp rises in 2005 and 2012.

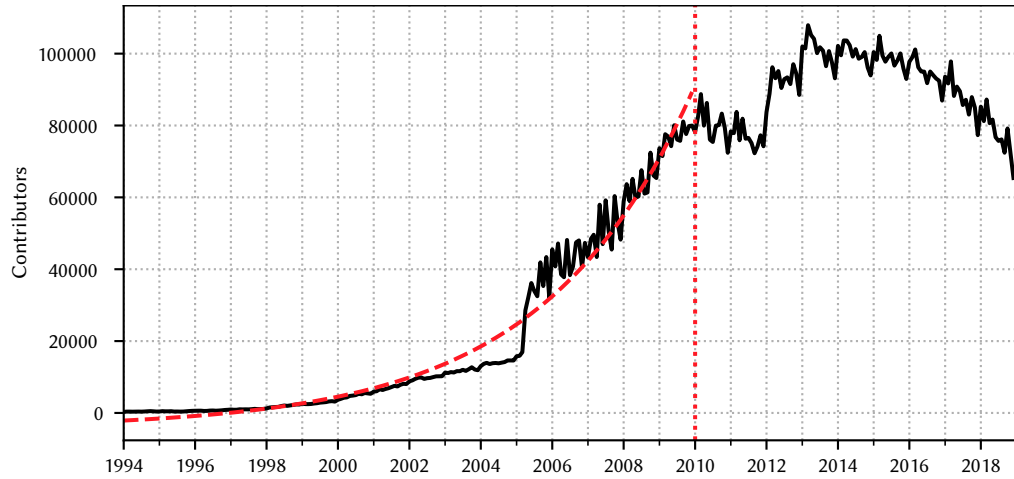


Fig. 7. Total contributors per month over time. The exponential curve does not fit after 2010. The number of contributors peaked in 2013, and since 2015 the general trend has been a decline.

Observation 8

We observe an erratic rise of contributors in 2005, and again in 2012.

↗ Figure 7

We cannot with certainty attribute the two sharp rises in contributors to any definite cause. However, it is possible that the factor mentioned as a source of measurement error in counting contributors (Section 3.4), namely an increase in bot activity, is responsible. In a sample of GitHub projects, [51] identified a significant increase in bot adoption, beginning at a slightly later date, after 2013. Our sample includes but is not limited to GitHub projects, which might explain the difference. Other possible factors are the increase of paid developers in this period [36], potentially with multiple professional affiliations, and the widespread shift from centralized to distributed version control (git was published in 2005), which changed the way in which commits were attributed [39].

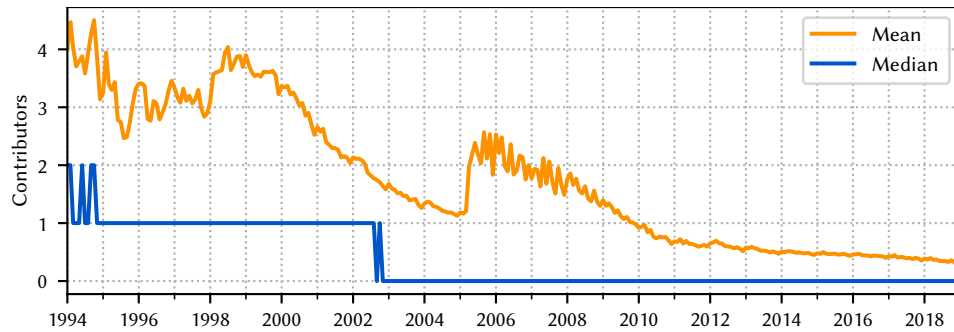


Fig. 8. Mean and median contributors per project per month. The mean of monthly commits has declined steadily since 2016, and the median reached and has remained at zero since 2003.

Observation 9

The average number of monthly contributors decreases over time.

↗ Figure 8

The median of contributors per month reaching zero means that more than half of the projects have no monthly contributors. Also the mean has a clear downwards trend.

As elaborated in Section 3.4 on systematic errors of measuring contributors, the exact numbers must be treated with caution. Especially tracking individual contributors, and matching identities to persons are error-prone and became increasingly difficult with distributed version control systems such as git. However, we assume that the number of identities per user remains stable over time: Most open source developers want to be identifiable and rewarded for their contributions. Both trends, the number of contributors and the effort spent in open source (measured in commits per contributor) are downwards. The reduction in effort per person is consistent with research describing the growing recognition of episodic participation in open source projects [4].

6 LIMITATIONS

The most serious threat to our the validity of our study is the unknown precision and accuracy of Open Hub as measurement system. This could result both in a sample of open source projects, that is not representative towards the universe of all open source projects and insufficient precision and accuracy of the detailed measurements. In a first attempt, we evaluated the precision and accuracy measurements by Open Hub by manual measuring lines of code and commits on five projects based on maximum variation sample: very large projects (Linux Kernel, OpenStack), mid-sized projects (KDE, Golang), and small projects (BeeTee). We found no significant deviations beyond the crawling jitter. However, this cannot be seen as a full evaluation as it is not statistically sound. If Open Hub as measurement system is faulty, this also affects [32] and, of course, Study C [12].

[5] assessed the quality of Ohloh in its version of the year 2013. They observed improper (1) SVN configurations, (2) missing and inconsistent (3) values. We also encountered the issue of projects which cannot be crawled due to outdated or wrong repository configuration (which we excluded). We also encountered similar problems while crawling Open Hub: 253,007 of 477,851 open source do not contain any information on the development activity and 881 projects had negative sizes, which we therefore excluded. We also found that the accumulated number lines added does not fit to the measured lines of code.

Open Hub marks reported duplicates of open source projects (mostly Linux kernel). How Open Hub detects duplications remains unclear. We assume further, undetected duplications. With excluding duplicates of the largest open source projects (e.g. Linux kernel), we believe that we excluded a significant, though not complete set of duplicates. In open source, forking is a common phenomenon. A fork is a secession from an existing open source project to a new, independent development. This split applies not only to the software development itself, but also to the developer community. The data set includes forks such as LibreOffice (forked from OpenOffice), but does not contain temporarily forks such those from GitHub. Although it can be extended, the current version of the novel open source life-cycle model does not cover the practice of forking. Although we excluded duplicated open source projects from our analysis, we relied on the duplication detection by Open Hub, which is not publicly available.

Obviously the choice of data source can affect findings. A recent study evaluating the Software Heritage Archive found public software growth rates to be exponential over a period of more than 40 years [40]. The difference in findings

can be potentially be explained in two ways which do not challenge the integrity of either data source. First, Open Hub includes only collaborative projects under an open source license, while The Software Heritage Archive preserves all public code, including, for instance, example code, personal websites hosted on GitHub, in addition to non-code materials, such as collaborative writing projects hosted on GitHub. We assume that Software Heritage is a superset of open source, but the stake of open source is unknown. Second, Open Hub explicitly seeks to exclude duplicates, while the aforementioned study found that files and commits in multiple contexts—in other words, duplication—was a significant factor in the increase of source code. The same is true for GitHub, as of today one of the largest code hosting platforms and the related GHTorrent project [20]. An unknown number of projects on GitHub is not intended to be an open source project aligning with the definition of open source and having no open source license.

We have good reasons to believe that the Open Hub measurement system is appropriated and applicable:

- Other studies have also relied on Open Hub and its predecessor Ohloh [32] assuming representativeness of Open Hub for open source – including one of the studies replicated [12].
- The universe contains 477,851 open source projects (in January 2020) and is thus the largest collection of exclusively open source projects we are aware of.
- Different platforms (such as GitHub, Gitlab, BitBucket, etc.) and tools are covered.
- We assume the data set tend to be more towards large, vivid, and larger open source projects, and to neglect code dumps, non-code open source projects, and intentionally hidden open source projects.

All measurements are based on the assumption that a human being contributed to open source. However, a contributor does not need to be human being and a commit or line of code is not necessarily written by a human being. For example, at Google most of the commits come from bots [35]. The extent of bot-generated code in open source is unknown. Bots do not affect the life-cycle (as long as bots are triggered by outside events and are not able to kick off open source projects), but all other measurements are affected.

The number of rejected or abandoned contributions is unknown and not considered in this work. However, because open source became a mass phenomenon, the acceptance rate may not be constant, but decreasing.

In our study, we refer to deleted projects when we actually mean not reachable for Open Hub (any more). This could also be the case for moved or archived projects or using a new version control system and not updating the Open Hub profile.

Changing the version control system (e.g. Linux kernel started in 1991, but switched to *git* in April 2005) can disturb the results. A change of version control system from centralized to distributed can also affect the perceived number of contributors because in earlier systems, commits were often attributed to the committer, regardless of who actually authored the commit [39].

7 DISCUSSION

We are surprised by our findings, and believe that a reproduction study utilizing a different data source is necessary to draw conclusions. There are two possible reasons for our results: One explanation is a limitation of the data source. The other option is that the data are correct, and open source has reached a (temporal) plateau. We discuss each of these possibilities in more depth below.

If Open Hub is not a representative data source, this could be due to inherent flaws, which we have discussed in Section 6 along with the steps we have taken to ameliorate this concern. Another possibility is that open source is

shifting from larger, centralized projects of the type tracked by Open Hub to smaller, more distributed projects. This would mean that growth in open source simply is not captured in a curated data source such as Open Hub.

In the case of the findings being an accurate reflection of the state of open source, future research is needed to explain this change, as companies and open source projects will need to adopt strategies which address the limited resources. While the data we investigated does not provide an answer, we can think of a number of potential explanations:

- A decrease in developers willing to volunteer, and no corresponding increase in paid development work
- The shift from volunteer to paid contributions reducing the effective time for contributing for each participant, due to company resource management
- An increase in episodic participation [3], with more people preferring to volunteer less
- A generational shift (the mean age of contributors in 2005 was 31, and in 2017 it was 30 [17, 18]) from collective to reflexive volunteering [22], perhaps in response to the growing role of open source participation in career development
- Increasing code complexity requiring skills fewer developers possess, and discouraging newcomers [47]
- Increasing formalization of software projects, requiring significant effort on the part of developers to adhere to submission or foundation guidelines, similar to what has been observed with Wikipedia [24]
- A decreased quality of contributions and, therefore, a lower acceptance rate of contributions and an overload for reviewers and committers

The data is not sufficient for any prediction model on the growth of open source: On the one hand, we could not overcome concerns towards the measurement accuracy and precision of our measurement system Open Hub (Observations 6 and 8, unclear project duplication detection, unknown bot activity) and unclear representativeness of our data set. On the other hand, open source could have reached a local or global maximum, the growth could be best described as bell-shaped or does not follow any regular pattern.

8 CONCLUSION

In this study, we conducted a large-scale sample study on open source projects and their cumulative growth. We replicated and reproduced measurements on lines of code, commits, and projects from prior studies and added a fourth measurement: contributors. We leveraged Open Hub as measuring system to measure development activities of 224,342 open source projects over the last 25 years with respect to those four quantities.

We could confirm an initial, transient exponential growth as claimed by Study C [12]. However, none of our accumulated measurements on lines of code, commits, contributors, or number of projects remained exponential, quadratic, or linear in terms of growth as suggested by the prior studies. In fact, we already passed the peak and observed a downwards trend for the measurements on commits and contributors in our data set.

Still the greatest weakness of our study is the same as for the replicated studies: the underlying data source – in our case Open Hub. Although we have good reasons to believe that Open Hub is a suitable data source, it is assumed to be representative in other studies [12, 32], and the alternative data sets available have their own limitations, we are still stuck for an answer if the open source project universe observed by Open Hub is a representative sample for open source or not. Thus, we encourage other researchers to reproduce our study. We published all analysis scripts set up in Jupyter notebooks and the full data set in raw and preprocessed state to the extent the data license allows.

Replications with alternative data sets will further narrow down the question raised in this paper: Quo vadis, open source?

ACKNOWLEDGMENTS

We thank Julian Frattini for his comments that greatly improved the manuscript and Peter Degen-Portnoy for his support during the data collection. This work was supported by the KKS Foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

REFERENCES

- [1] Abdulkareem Alali, Huzefa Kagdi, and J.I. Maletic. 2008. What’s a Typical Commit? A Characterization of Open Source Software Repositories. In *2008 16th IEEE International Conference on Program Comprehension*. IEEE, 182–191. <https://doi.org/10.1109/ICPC.2008.24>
- [2] Oliver Arafat and Dirk Riehle. 2009. The Commit Size Distribution of Open Source Software. In *2009 42nd Hawaii International Conference on System Sciences*. IEEE, 1–8. <https://doi.org/10.1109/HICSS.2009.421>
- [3] Ann Barcomb, Andreas Kaufmann, Dirk Riehle, Klaas-Jan Stol, and Brian Fitzgerald. 2018. Uncovering the Periphery: A Qualitative Survey of Episodic Volunteering in Free/Libre and Open Source Software Communities. *IEEE Transactions on Software Engineering* (2018), 19. <https://doi.org/10.1109/TSE.2018.2872713>
- [4] Ann Barcomb, Klaas-Jan Stol, Brian Fitzgerald, and Dirk Riehle. 2020. Managing Episodic Volunteers in Free/Libre/Open Source Software Communities. *IEEE Transactions on Software Engineering* 5589, November 2019 (2020), 1–1. <https://doi.org/10.1109/TSE.2020.2985093>
- [5] Magiel Bruntink. 2014. An initial quality analysis of the Ohloh software evolution data. *Electronic Communications of the EASST* 65 (2014). <https://doi.org/10.14279/tuj.eceasst.0.906.889>
- [6] Andrea Capiluppi and Daniel Izquierdo-Cortázar. 2013. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empirical Software Engineering* 18, 1 (feb 2013), 60–88. <https://doi.org/10.1007/s10664-011-9191-7>
- [7] Andrea Capiluppi, Patricia Lago, and Maurizio Morisio. 2003. Characteristics of open source projects. *Seventh European Conference on Software Maintenance and Reengineering 2003 Proceedings* (2003), 317–327. <https://doi.org/10.1109/CSMR.2003.1192440>
- [8] Andrea Capiluppi and Martin Michlmayr. 2007. From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects. In *Open Source Development, Adoption and Innovation*. Vol. 234. Springer US, Boston, MA, 31–44. https://doi.org/10.1007/978-0-387-72486-7_3
- [9] Maximilian Capraro, Michael Dorner, and Dirk Riehle. 2018. The patch-flow method for measuring inner source collaboration. In *Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18*. ACM Press, New York, New York, USA, 515–525. <https://doi.org/10.1145/3196398.3196417>
- [10] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2012. Free/Libre open-source software development. *Comput. Surveys* 44, 2 (2012), 1–35. <https://doi.org/10.1145/2089125.2089127>
- [11] Carlo Daffara. 2007. Estimating the number of active and stable floss projects. <https://robertogaloppini.net/2007/08/23/estimating-the-number-of-active-and-stable-floss-projects/>
- [12] Amit Deshpande and Dirk Riehle. 2008. The Total Growth of Open Source. In *Open Source Development, Communities and Quality*. Vol. 275. Springer US, Boston, MA, 197–209. https://doi.org/10.1007/978-0-387-09684-1_16
- [13] Gabriel Farah, Juan Sebastian Tejada, and Dario Correal. 2014. OpenHub: a scalable architecture for the analysis of software quality attributes. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, New York, New York, USA, 420–423. <https://doi.org/10.1145/2597073.2597135>
- [14] Joseph Feller and Brian Fitzgerald. 2002. *Understanding Open Source Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. <https://doi.org/10.5555/513726>
- [15] Apache Foundation. 2019. *The Apache Incubator*. <http://incubator.apache.org>
- [16] Eclipse Foundation. 2019. *Development Process Project Lifecycle*. https://www.eclipse.org/projects/dev_process/#6_2_Project_Lifecycle
- [17] Rishab Ghosh and Rüdiger Glott. 2005. *FLOSSPOLs: Skills Survey Interim Report*. Technical Report. http://flosspols.merit.unu.edu/deliverables/D10HTML/FLOSSPOLs-D10-skills%20survey_interim_report-revision-FINAL.html
- [18] Github. 2017. The Open Source Survey. <http://opensourcesurvey.org/2017/> Accessed 11 Nov 2018.
- [19] Michael Godfrey and Qiang Tu. 2004. Growth, evolution, and structural change in open source software. (2004), 103. <https://doi.org/10.1145/602461.602482>
- [20] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) (MSR ’13). IEEE Press, Piscataway, NJ, USA, 233–236. <http://dl.acm.org/citation.cfm?id=2487085.2487132>
- [21] Lile P. Hattori and Michele Lanza. 2008. On the nature of commits. *Aramis 2008 - 1st International Workshop on Automated engineering of Autonomous and runtime evolving Systems, and ASE2008 the 23rd IEEE/ACM Int. Conf. Automated Software Engineering* (2008), 63–71. <https://doi.org/10.1109/ASEW.2008.4686322>
- [22] Lesley Hustinx and Frans Lammertyn. 2003. Collective and reflexive styles of volunteering: A sociological modernization perspective. *Voluntas: International Journal of Voluntary and Nonprofit Organizations* 14, 2 (2003), 167–187.
- [23] ISO/IEC. [n.d.]. *International Vocabulary of Metrology—Basic and General Concepts and Associated Terms*. Technical Report.

- [24] Nicolas Jullien, Kevin Crowston, and Felipe Ortega. 2015. The Rise and Fall of an Online Project. Is Bureaucracy Killing Efficiency in Open Knowledge Production? In *Proceedings of the 11th International Symposium on Open Collaboration (OpenSym 2015)*. ACM, New York, NY, USA.
- [25] Stefan Koch. 2004. Profiling an Open Source Project Ecology and Its Programmers. *Electronic Markets* 14, 2 (jun 2004), 77–88. <https://doi.org/10.1080/10196780410001675031>
- [26] Stefan Koch. 2007. Software evolution in open source projects—a large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice* 19, 6 (nov 2007), 361–382. <https://doi.org/10.1002/smr.348>
- [27] Carsten Kolassa, Dirk Riehle, and Michel A. Salim. 2013. A Model of the Commit Size Distribution of Open Source. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7741 LNCS. 52–66. https://doi.org/10.1007/978-3-642-35843-2_6
- [28] Carsten Kolassa, Dirk Riehle, and Michel A. Salim. 2013. The empirical commit frequency distribution of open source projects. In *Proceedings of the 9th International Symposium on Open Collaboration - WikiSym '13*. ACM Press, New York, New York, USA, 1–8. <https://doi.org/10.1145/2491055.2491073>
- [29] Yoshitaka Kuwata and Hiroshi Miura. 2015. A study on growth model of OSS projects to estimate the stage of lifecycle. *Procedia Computer Science* 60, 1 (2015), 1004–1013. <https://doi.org/10.1016/j.procs.2015.08.142>
- [30] Sihai Lin, Yutao Ma, and Jianxun Chen. 2013. Empirical evidence on developer’s commit activity for open-source software projects. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE 2013-Janua*, January (2013), 455–460.
- [31] Yutao Ma, Yang Wu, and Youwei Xu. 2014. Dynamics of open-source software developer’s commit behavior. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*. ACM Press, New York, New York, USA, 1171–1173. <https://doi.org/10.1145/2554850.2555079>
- [32] Álvaro Menezes and Rafael Prikladnicki. 2018. Diversity in software engineering. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering - CHASE '18*. ACM Press, New York, New York, USA, 45–48. <https://doi.org/10.1145/3195836.3195857>
- [33] Siobhan O’Mahony and Joel West. 2005. What makes a project open source? Migrating from organic to synthetic communities. *Academy of Management conference, Technology and Innovation Management division, Honolulu, August 2005* (2005), 39.
- [34] Hans E. Plesser. 2018. Reproducibility vs. Replicability: A Brief History of a Confused Terminology. *Frontiers in Neuroinformatics* 11, January (jan 2018), 1–4. <https://doi.org/10.3389/fninf.2017.00076>
- [35] Rachel Potvin and Josh Levenberg. 2016. Why Google stores billions of lines of code in a single repository. *Commun. ACM* 59, 7 (jun 2016), 78–87. <https://doi.org/10.1145/2854146>
- [36] Dirk Riehle, Philipp Riemer, Carsten Kolassa, and Michael Schmidt. 2014. Paid vs. Volunteer Work in Open Source. In *2014 47th Hawaii International Conference on System Sciences*. IEEE, 3286–3295. <https://doi.org/10.1109/HICSS.2014.407>
- [37] Gregorio Robles, Juan Jose Amor, Jesús M. González-Barahona, and Israel Herraiz. 2005. Evolution and Growth in Large Libre Software Projects. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, Vol. 2005. IEEE, 165–174. <https://doi.org/10.1109/IWPSE.2005.17>
- [38] Gregorio Robles, Jesús M. González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. 2014. Estimating development effort in Free/Open source software projects by mining software repositories: a case study of OpenStack. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*. ACM Press, New York, New York, USA, 222–231. <https://doi.org/10.1145/2597073.2597107>
- [39] Christian Rodriguez-Bustos and Jairo Aponte. 2012. How Distributed Version Control Systems impact open source software projects. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 36–39. <https://doi.org/10.1109/MSR.2012.6224297>
- [40] Guillaume Rousseau, Roberto Di Cosmo, and Stefano Zacchiroli. 2019. Growth and Duplication of Public Source Code over Time: Provenance Tracking at Scale. *arXiv:1906.08076v1*.
- [41] Chanchal Kumar Roy and James R Cordy. 2006. Evaluating the evolution of small scale open source software systems. *Special issue on CIC 2006, 15th International Conference on Computing, Research in Computing Science* 23 (2006). <https://doi.org/10.1.1.61.8405>
- [42] Munish Saini and Kuljit Kaur. 2015. A Review of Software Development Life Cycle Models. *International Journal of Software Engineering and Its Applications* 8, 3 (2015), 417–434. <https://doi.org/10.14257/ijseia.2014.8.3.38>
- [43] Charles Schweik. 2003. The Institutional Design of Open Source Programming: Implications for Addressing Complex Public Policy and Management Problems. *First Monday* 8, 1 (jan 2003). <https://doi.org/10.5210/fm.v8i1.1019>
- [44] Maha Shaikh and Tony Cornford. 2009. Innovating with open-sourcing: Governance concerns for managers. *15th Americas Conference on Information Systems 2009, AMCIS 2009* 4 (2009), 2407–2416.
- [45] Andrej-Nikolai Spiess and Natalie Neumeyer. 2010. An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. *BMC Pharmacology* 10, 1 (dec 2010), 6. <https://doi.org/10.1186/1471-2210-10-6>
- [46] Megan Squire. 2017. The Lives and Deaths of Open Source Code Forges. In *Proceedings of the 13th International Symposium on Open Collaboration - OpenSym '17*. ACM Press, New York, New York, USA, 1–8. <https://doi.org/10.1145/3125433.3125468>
- [47] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.
- [48] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of Software Engineering Research. *ACM Transactions on Software Engineering and Methodology* 27, 3 (sep 2018), 1–51. <https://doi.org/10.1145/3241743>
- [49] Giancarlo Succi, James Paulson, and Armin Eberlein. 2001. Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products. in *Proceedings of the Workshop on Economics-Driven Software Engineering Research, Edser 3* (2001), 14–15. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.8888>

- [50] Masateru Tsunoda, Akito Monden, Takeshi Kakimoto, Yasutaka Kamei, and Ken-ichi Matsumoto. 2006. Analyzing OSS developers' working time using mailing lists archives. In *Proceedings of the 2006 international workshop on Mining software repositories - MSR '06*. ACM Press, New York, New York, USA, 181. <https://doi.org/10.1145/1137983.1138031>
- [51] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco Aurelio Gerosa. 2018. The Power of Bots. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (nov 2018), 1–19. <https://doi.org/10.1145/3274451>
- [52] Joel West and Siobhán O'Mahony. 2008. The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation* 15, 2 (apr 2008), 145–168. <https://doi.org/10.1080/13662710801970142>
- [53] Donald E. Wynn. 2004. Organizational Structure of Open Source Projects: A Life Cycle Approach. *SAIS 2004 Proceedings* 47, 6 (may 2004). <https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1046&context=sais2004>