Acta Univ. Sapientiae, Informatica, **16, 2** (2024) 236–254

DOI: 10.47745/ausi-2024-0013

A large-scale analysis of production effort changes in software projects

Kristóf Szabados
Eötvös Loránd University
Budapest, Hungary
SzabadosKristf@gmail.com

Abstract.

Although software is essential to modern society, its development process remains poorly understood. It is easy to find tools/techniques/methods promoted to help reach ever-higher production volumes and great improvements, even though this contrasts previous academic observations.

This study aims to expand on previous research regarding the evolution of software development efforts by analyzing a large dataset of contemporary open-source projects.

We examined 875 popular and large GitHub repositories and found a strong correlation between accumulated effort and quadratic functions of time, in 73% of projects, with linear models applicable in 41%. Our analysis of long-term trends suggests that no major external events (e.g., economic downturns, technological shifts) have impacted significantly all projects over the past 25 years.

These findings challenge the perception of rapid, exponential improvements in software development efficiency. While further research is needed, our results suggest a disconnect between software development's perceived and empirical realities.

Key words and phrases: Empirical Study, Large-Scale, Software Evolution, Longitudinal, Conway's Law, Mirroring Law, Lehman's Laws, Software Development Tools, Software Engineering Laws, Sustainability, Cost Reduction

1 Introduction

Software is ubiquitous in modern society. It helps us navigate, communicate, and manage energy resources. It drives companies, trades on markets, and supports healthcare.

Developing large-scale software systems often takes years and is an effort-intensive endeavour. Accelerating software development could have a positive impact on both economic growth and social well-being. Improving efficiency could improve results, reduce costs and free up resources for other activities.

Given the decades-long history of software development, we have access to a wealth of empirical data. By analysing the source code repositories of open-source projects, we can gain valuable insights into the factors that influence development efforts. This will ensure that upcoming development efforts are based on a solid foundation of evidence-based practices and strategies instead of assumptions and speculations. By examining historical trends, we can uncover hidden correlations and potential causal relationships between external factors and development efforts, ultimately enabling us to make more informed decisions and optimise our development processes.

In this paper, we report on our study on how the effort invested into software development changes over time, and how seemingly there was no event in the last 25 years that would have impacted all projects on its own, using 875 open-source projects.

This paper is organised as follows. Section 2 presents related works from the literature. Section 3 describes our work. Section 4 presents our results and analyses. Section 5 their validity. Finally, Section 6 shows our conclusions, and Section 7 offers ideas for further research.

2 Literature Review

In this section, we present earlier literature related to our work. While we view our work as a natural extension of [1] and [2] (presented in Section 2.5), we also present research results related to strong patterns observed in other aspects of software systems. We accept the potential influence of external factors, such as international laws, regulations, and business objectives, on software development practices. However, we do not wish to prioritize these factors in our literature review, as we lack empirical evidence to assess their relative importance.

In the first group of sections, factors external to the software: organisations intentionally design and govern the overall architecture of their products to achieve their business targets (Sections 2.1, 2.2). Followed by sections on the generality and inevitability of the internal structure of large software systems (Section 2.3), all software systems evolving similar size distributions (Section 2.4). Finally, previous works on how all software systems evolve in similarly predictable ways (Section 2.5).

2.1 The impact of organisational factors on software systems

Empirical observations have identified a strong relationship between an organisation's communication and product structure [3], [4]. In 2008 Nagappan et al. [5] showed that organisational metrics predict failure-proneness better than code complexity, coverage, internal dependencies, churn and pre-release bug measures. By 2022 it was used by firms as a superior strategy [6] to maximise business benefits [7]. This indicates the importance of the legal and business environment over any technical considerations.

Following these laws, contemporary System Architects consider among others Taxation [8], Export control [9], and Geopolitics [10] when planning software architecture and the organisation developing it. Contemporary Project Management recommends [11] tailoring a selected development approach first for the organisation and second to the project.

2.2 The impact of Project Management on software Projects

Researchers have identified [12]–[14] that Project Management[1] techniques and processes are the critical factors contributing to project success.

Empirical observers have noted [16] that 94% of troubles and possibilities for improvement are the responsibility of management. Recommended preventing problems with systemic problem-solving performed with scientific rigour [17]–[19] instead of celebrating solving crises and heroes putting out fires. Understanding, that writing programs "is only a small part of Software Engineering" ([20]). This indicates the importance of intentional and professional management.

2.3 The dependency networks of software systems

Empirical researchers have shown that several architectural properties of software systems are scale-free[2], like many real-life networks. Class collaboration graphs of the C++ language [21], Class, method, and package collaboration graphs of Java [22], [23], connections between the modules in TTCN-3[3] [24], [25], file inclusion graphs in C [26], and the runtime object graph of most of the Object Oriented Programming languages in general [27], the relationships of distributed software packages [28], [29] show scale-free properties.

[1] "Project Management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements" [15].

[2] A network is called scale-free, when its degree distribution, follows a power law.

[3] TTCN-3 is the abbreviation of Testing and Test Control Notation Version 3

Taube-Schock et al. [30] showed, that approximately scale-free structures should arise for both between-module and overall connectivity from the preferential attachment-based models, not as the result of poor design. That high coupling is unavoidable and might even be necessary for good design, contradicting previous ideas about software structure, particularly the "high cohesion/low coupling" maxim. This indicates that software design is not done before development, but emerges automatically during it, with developers having limited control over it.

2.4 The size distribution of software systems

Empirical studies have revealed that several metrics correlate to the point of redundancy[4] [31]. Measuring Source Lines Of Code would be enough to obtain a landscape of the evolution of the size and complexity of FreeBSD [32].

Empirical researchers have shown logarithmic distributions in various places: module lengths of IBM 360/370 and PL/S code [33], Java class sizes [34], tokens in Java [35], [36], all metrics measured on FreeBSD [32], for five (C, C++, Java, Python, Lisp) of the top seven programming languages used in the Linux code [37] and LOC in Smart Contracts for the Ethereum blockchain [38]. Stating that "whatever is measured in a large scale system" logarithmic distribution is observed in most cases [39].

Hatton used [40], [41] the Conservation of Hartley-Shannon Information to show strong evidence for unusually long components being an inevitable by-product of the total size of the system, validating the claims on 100 million lines of code in 7 programming languages and 24 Fortran 90 packages. Highlighting the importance of changing software design techniques, from attempting to avoid the essentially unavoidable to mitigating its damaging effects.

2.5 The evolution of software systems

Since Lehman published the laws of software evolution [42], plenty of empirical research [1], [2], [43]–[50] show that the laws seem to be supported by solid evidence. To the point that the gross growth trends can be predicted by a mean absolute error of order 6%[1], [2], [51], [52].

Looking at the impact of outside effects on software growth, empirical researchers observed [1] that "the introduction of continuous integration, the existence of tool support for quality improvements in itself, changing the development methodolo-

[4]Cyclomatic Complexity, the Number of Lines of Code, Statements, Classes, Files, public APIs, and public undocumented APIs are redundant metrics, with Cyclomatic Complexity in classes and functions measuring the same subject

gies (from waterfall to agile), changing technical and line management structure and personnel caused no measurable change in the trends of the observed Code Smells", on industrial Java and C++ projects [2], that changing architects, going open-source or the organisation moving to a different building had no easily discernible effect on development.

The works performed on large open-source systems [2], [44], [46]–[49], [52] serve as observations supporting the understanding that there might not have been any hardware, software, tooling, methodological, social, or other change at least since 2000, that would have significantly changed the development speed of large software systems already started.

3 Methodology

This section presents our aim and work in technical terms.

We wanted to extend the knowledge available on how software systems evolve with up-to-date information on a dataset of large, diverse, and long-running projects.

We selected GitHub as the data source, one of the largest sites hosting software project repositories. To ensure language independence, we focused on change counts reported by Git for each merge. This approach allowed us to treat all changes, including code, comments, documentation, test examples, and other text artefacts equally.

We also decided to treat deletions and insertions as equally significant changes, valuing careful removal of unnecessary code at the same level as adding new ones.

The following metrics were tracked:

- *Cumulative Commits*: The total number of commits contributed over time.
- *Cumulative Effort*: The total number of insertions and deletions made over time.
- *Lines*: The net change in lines of text, calculated by subtracting deleted lines from inserted lines.

The process for gathering and processing the data:

1. *Selection*: We identified the 100 most liked/stared GitHub repositories for each programming language[5] and selected those with over 3000 commits by mid-2024.

[5]The list of most popular repositories by programming languages we used is shown on GitHub-Ranking

2. *Repository filtering*: We excluded repositories triggering virus detection (e.g., Metasploit Framework), from this work, so that we would not compromise the security of our devices and the networks they have to connect to.

3. *Log extraction*: We extracted Git logs from the main branch of each repository. With the adjusted settings:

   (a) Increased *diff.renamelimit* to 130,000.

   (b) Used --first-parent, for branch history analysis[6].

   (c) We used the commit date (”%cd”). We observed the author date as easily manipulated[7].

4. *Data cleaning, filtering*: The data were cleaned up and filtered.

5. *Metric tracking*: Changes in the number of commits, lines, and effort values were tracked.

6. *Data Analysis*: The data collected was analysed.

Data cleaning and filtering involved:

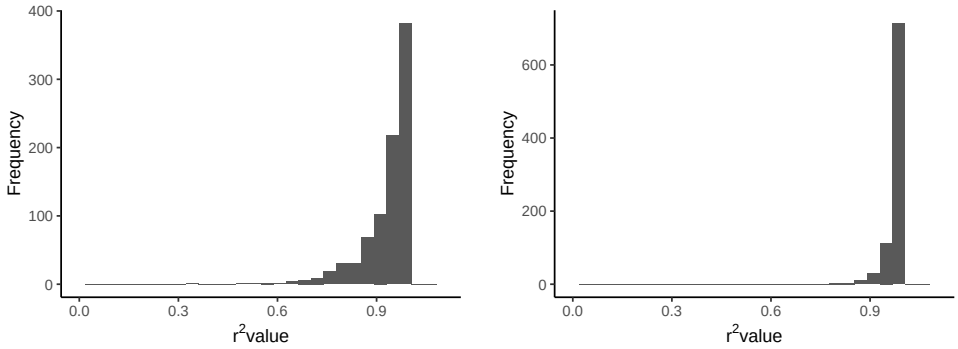
- In some projects the number of lines dropped to near zero before resuming growth. When these commits had comments hinting at restarting the project[8] we split the project into two projects at those commits.

- We removed commits if there were only a few and much earlier than the actual work started[9].

- We investigated every commit that made the repository look like having negative content and modified their date to that of their later parent[10].

4 Results and discussion

This section presents our results and analyses.



(a) (b)

Figure 1: r^2 value distribution for commits with linear 1(a) and quadratic 1(b) fitting

4.1 General overview

To investigate the relationship between accumulated commits, effort and project development, we fitted linear and simple quadratic regression models to the data. Figure 1(a) shows the distribution of r^2 values when matching the accumulated number of commits with a linear model. Of the 875 projects, 249 match above 0.98, 495 above 0.95 and 687 above 0.9. Figure 1(b) shows the same data matched against a quadratic model, with 586 matches above 0.98, 782 above 0.95 and 850 above 0.9.

Figure 2(a) shows the distribution of r^2 values when matching the accumulated effort values with a linear model of the 875 projects 126 match above 0.98, 357 above 0.95 and 576 above 0.9. Figure 2(b) shows the same data matched against a quadratic model, 344 matches above 0.98, 636 above 0.95 and 789 above 0.9.

This indicates that software development mostly follows clear evolution patterns. While these models may not achieve the highest precision, they might offer a practical and effective approach for industrial applications where precise forecasting is not always critical. For most projects, simple linear or quadratic models provide a reasonable fit, suggesting that project evolution can be approximated with functions that change direction at most once.

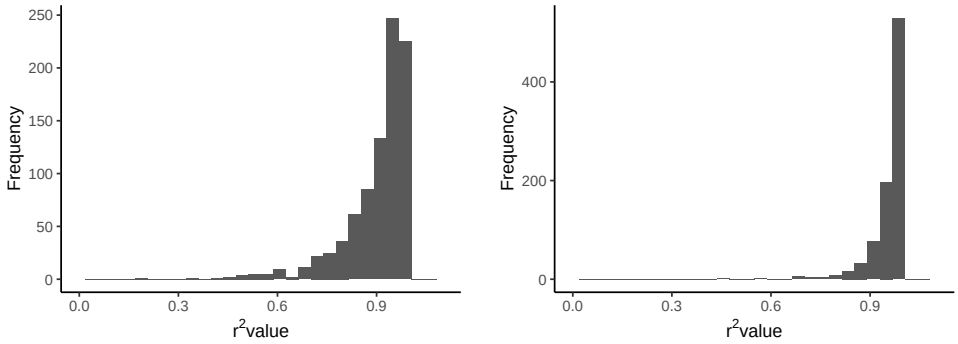
[6]This also simplified handling octopus merges. Linux has merges with up to 66 parents, for example: commit hash

[7]Linux seems to have commits from 1970.01.01 and 2085.06.18.

[8]In tldraw, on 2023.04.21 everything is deleted, and re-created for a new version on 2023.04.25.

[9]Go has a "Hello World" commit in 1972, FFMPeg some commit months before the "Initial revision"

[10]In Ansible the first commit, by date, seems to delete 12 lines from an empty repository, but it builds on a commit from 2012.02.24.



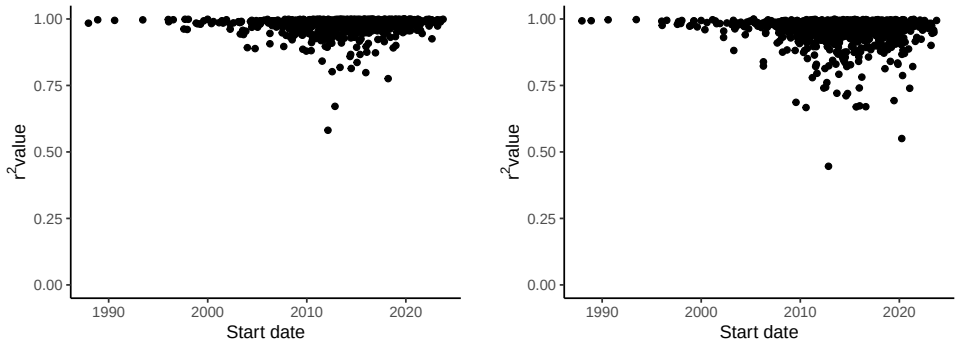
(a) (b)

Figure 2: r^2 value distribution for effort with linear 2(a) and quadratic 2(b) fitting

4.2 Quality in time

When investigating how the r^2 values might depend on when the projects started, we observed that newer projects tend to exhibit worse fits to linear or quadratic models (see Figure 3).

It is important to note that a selection bias may skew this analysis. Many projects initiated after 2020 might not have yet reached the popularity levels or commit counts required for inclusion in our dataset.



(a) (b)

Figure 3: r^2 value distribution for commits 3(a) and effort 3(b) when correlated with quadratic models, represented as a function of the starting time of the project

4.3 Observations using some of the oldest projects

Figure 4 shows how the accumulated effort invested into a project grows over time for some of the oldest projects.

While there were observable changes in the projects individually, it is clear that in the last 25 years, no external change had a significant, lasting impact on all projects.

Such external events include:

- Processor speed and core count grew by magnitudes.
- RAM size and throughput grew by magnitudes.
- Internal storage size and throughput grew by magnitudes while decreasing latency by magnitudes.
- Screen space to display information increased from around 640×480 to 1920×1080 and beyond.
- Graphical IDEs, code quality checking and refactoring, CI/CD and other productivity and support tools became available.
- The Internet became available, with several sites supporting developers.
- Cloud computing made vast amounts of resources easily reachable.
- Various methodologies were promoted to improve developer productivity and reduce defects: Scrum, Kanban, SAFe, LeSS, Nexus, Scrum@Scale, Agile@Scale, Lean, XP, FDD, AIDD, DSDM, UP, Six Sigma, DevOps and more.
- Computer Science extended understanding of how software systems evolve, how structure and distribution patterns emerge (Section 2).
- The FLoyd-Hoare logic was extended for parallel programs.
- A financial crisis, a global pandemic and a change in R&D TAX law[11].
- Some open-source projects accumulated thousands of contributors[12].
- AI improved, automated planning and decision-making, navigation in physical and abstract spaces[13]. ChatGPT.

[11] Section 174 in 2022.
[12] Linux has 15,700+, CPython 2,811, GCC 983 contributors according to GitHub.
[13] Already in 2001 demonstrating cooperative and competitive team level behaviour[53]

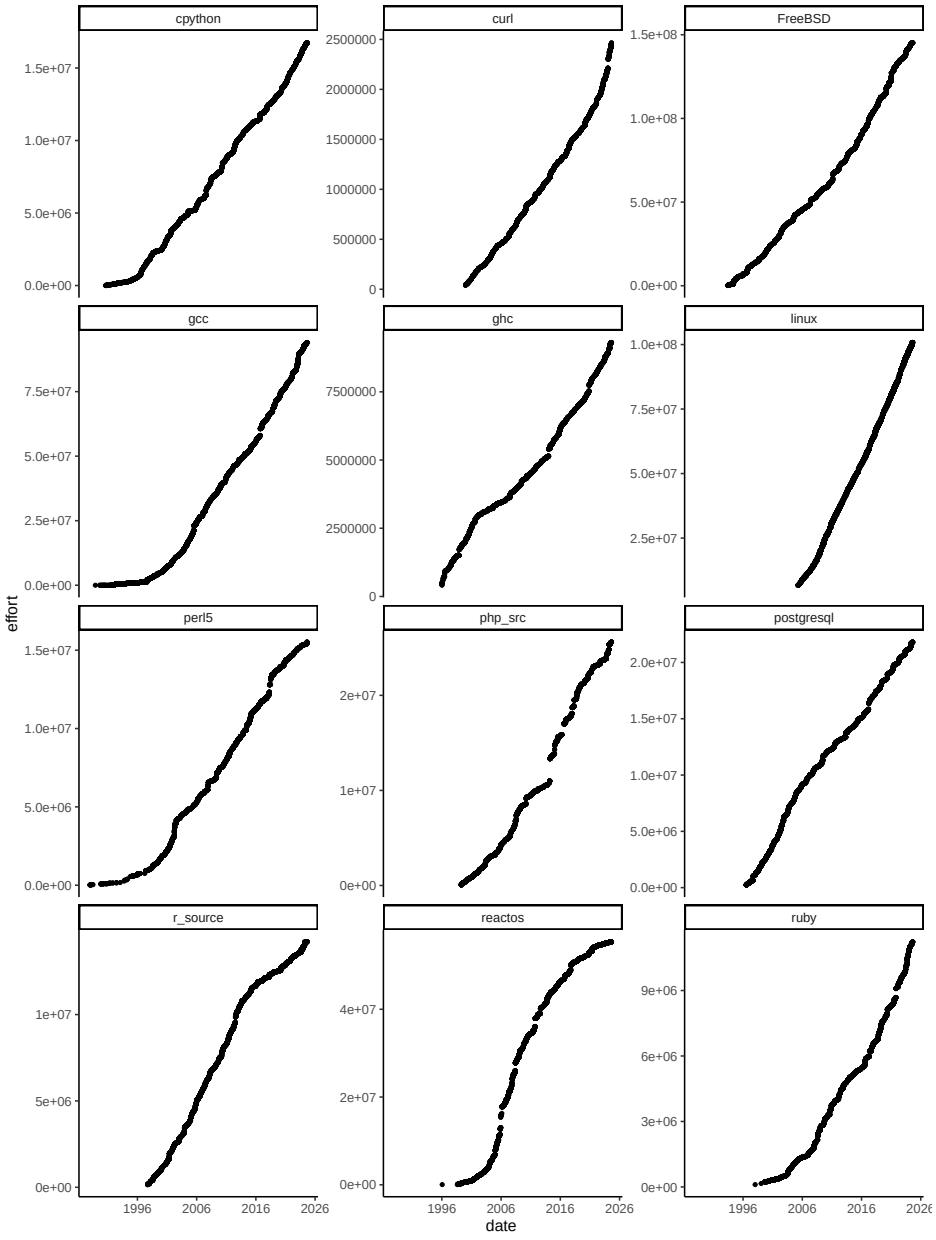


Figure 4: The effort graphs for some projects started before 2000.

According to our data, none of these improvements and events have impacted all projects at the time, on their own.

Other less generic events could include security patches, operating systems, programming languages, and toolset version updates. These are usually forbidden within shorter/smaller projects, and in longer projects managed closely as a parallel side-project to ensure no noticeable impact on the main project.

4.4 Projects with close to linear Effort trends are present in many programming languages

Our analysis (Figure 2) identified several projects demonstrating high r^2 values fitting the accumulated Effort measurements to a linear model for different programming languages. This indicates that the effect is language agnostic.

Figure 5 shows: APISIX (79.7% Lua), Appsmith (67.2% typescript), android-oss (92.7% Kotlin), BookStack (89.2% PHP), Cats (100% Scala), CockroachDB (89.9% Go), ColossalAI (92.8% Python), DevDocs (79.4% Ruby), Dokku (57.4% Shell script), egui (98.5% Rust), Envoy (87.6% C++), Fastify (90.8% Javascript), osu! (100% C#), Linux (98.4% C), AppFlowy (54.6% Dart), Bitcoin (65.3% C++, 20.2% Python).

4.5 Observations on the least fitting projects

This section presents our observations on the least fitting projects.

Only two projects, "HoTT/book" ($r^2 = 0.32$) and 'SecLists' ($r^2 = 0.49$), showed linear fits below 0.5 for commit accumulation. "HoTT/book", started in 2013, had approx. 50% of its total commits between 2013.04.01 and 2013.09.10. "SecLists", established in 2012, had approx. "62%" of its commits between 2024.06.15 and 2024.09.15.

Investigating fits below 0.5 for effort accumulation, we found 8 projects: HoTT/-book 0.18 and 0.45 (linear and quadratic fit), Middleman 0.35 and 0.69, Moya 0.41 and 0.71, Caffe 0.44 and 0.72, Frontend 0.45 and 0.74, plotly.R 0.49 and 0.79, Dapr 0.495 and 0.69, static-analysis 0.498 and 0.74.

In some of these projects, significant changes were introduced only by a few commits which were also removed soon after. For example (see Figure 6):

- Caffe: On 2013.11.07 approx. 1.3 million lines were added, seemingly most of them removed on 2014.02.26 with commit deleting 1.4 million lines.
- Moya: On 2015.06.16 47, 698 lines were added, on 2015.09.12 45, 993 removed, on 2015.09.14 52, 323 added and on 2015.10.27 51, 124 removed.

To illustrate the impact of such changes, removing these specific commits the fit would improve to r^2 values of 0.79 and 0.94 for Caffe, 0.50 and 0.79 for Moya, 0.68

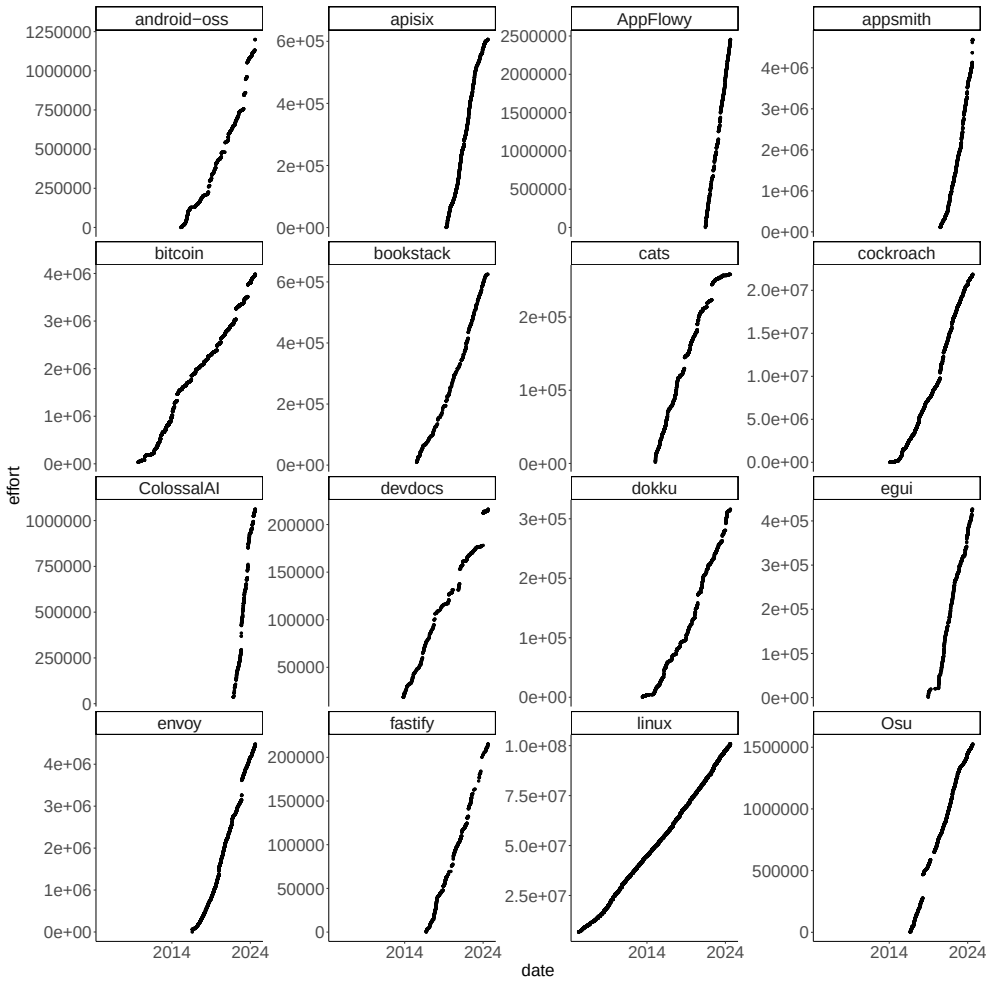
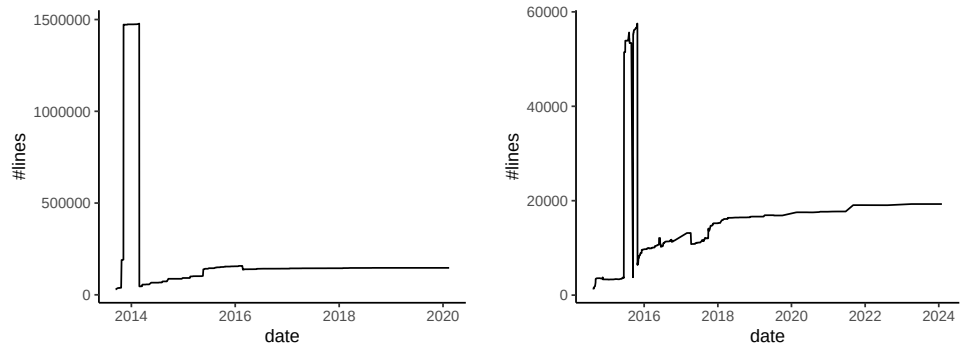


Figure 5: Projects with high r^2 values, developed in different programming languages

and 0.97 for Middleman, 0.83 and 0.98 for plotly.R, 0.67 and 0.74 for Dapr.

HoTT/book seems to have a large commit beside a large insertion and deletion pair on 2013.04.27. Removing only the insertion-deletion pair does not change the r^2 values significantly. The large insertion is a merge, indicating that a substantial amount of work is being done elsewhere.

Frontend seems to have many large temporal changes, Static-analysis underwent several changes between 2022.08.22 and 2022.09.24 that look like refactoring, mak-



(a) (b)

Figure 6: The number of lines for Caffe 6(a) and Moya 6(b)

ing them harder to analyse.

5 Threats to validity

This study might suffer from the usual threats to external validity. There might be limits to generalizing our results beyond our settings (the programming language used and repositories outside of GitHub).

One specific threat to validity might come from the fact that committers did modify the author and commit date of some of the commits. It could happen that there were other ways and methods to change commits, that we did not know about or notice. To address this threat we have to point out that most of the projects have several contributors (15, 700+ for Linux) and have been running for several years. We find it unlikely that an effort would exist to create misleading metrics on this scale.

A limit for generalizing our results might come from the selection criteria: popular, large, open-source projects present on GitHub. While GitHub is one of the largest sites hosting software project repositories, privately developed, less popular or smaller repositories may follow different trends.

6 Conclusion

This paper presents our study on how the effort invested into software development projects changes over time, using 875 popular and large open-source projects.

We presented earlier works on the structure of software systems and the external factors identified to impact them significantly. We also presented earlier works on the evolution tendencies of software systems, information that we are updating with contemporary data and extending in the number of projects.

In nearly 73% of the investigated projects, we found a strong correlation between accumulated effort and a quadratic function of time.

Despite significant advancements in areas related to software development over the past 25 years, our analysis of pre-2000 projects revealed no overarching external event that consistently impacted their development. Most projects seemed to be largely unaffected by external factors. This effect was seen regardless of the programming language used.

Our results show that software development seems to follow clear patterns, even if those are not intuitive. Our observations offer little hope for improving the speed of software development but indicate the possibility of improving efficiency by optimising headcount and investment into tooling ([54]).

Our findings support the prohibition of gold plating (any effort spent on achieving higher than necessary quality is likely to be taken away from other directly beneficial features), the consideration of outcome bias (as the outcome might not depend on the actions of individuals or decisions directly, it should be ignored when evaluating those decisions) and in general the delaying of decisions as long as feasible instead of action fallacy (since development seems to follow predictable trends it is not clear if decisions do have an impact on results or might only take up resources to make).

7 Further work

In the future, we plan to investigate more projects with low r^2 values. Further research could investigate the reasons behind the deviance we observed in some projects and explore for specific projects all events that happened with that project and their impact on it (e.g. [1]).

Further research could also investigate if there are differences in the effect based on the organisation performing the development or industrial area.

Further research could also investigate how and how far the headcount and tooling investment could be lowered while achieving the same results.

It is generally accepted that a low bus factor and the presence of the hero anti-pattern can harm projects. Our observations also seem to indicate that it is not just necessary to limit these effects, but also possible on long time horizons (in a 25 year time frame, people tend to leave projects). We leave it to further researchers to

investigate in detail, how the projects managed to keep these effects under control.

Data Availability: For our research we only used data publicly available on GitHub (the Git repositories of the projects). In every case, we used the latest version at the time of publication (pulling the latest changes for every project). We refer to specific phenomena in the article by providing direct links to the code version, using their unique commit hash identifier.

Acknowledgments: The authors thank Izabella Ingrid Farkas for her help and feedback on this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

[1] A. Kovacs and K. Szabados, "Internal quality evolution of a large test system–an industrial study," *Acta Univ. Sapientiae*, vol. 8, no. 2, pp. 216–240, 2016 (⇒ 237, 239, 249).

[2] A. Zsiga, "Termelékenységi trendek, minták elemzése szoftverfejlesztési projektekben," M.S. thesis, Eötvös Loránd University, 2019 (⇒ 237, 239, 240).

[3] M. E. Conway, "How do committees invent," *Datamation*, 1967 (⇒ 238).

[4] A. MacCormack, C. Baldwin, and J. Rusnak, "Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis," *Research Policy*, vol. 41, no. 8, pp. 1309–1324, 2012, ISSN: 0048-7333 (⇒ 238).

[5] N. Nagappan, B. Murphy, V. Basili, and N. Nagappan, "The influence of organizational structure on software quality: An empirical case study," Microsoft Research, Tech. Rep. MSR-TR-2008-11, Jan. 2008, p. 11 (⇒ 238).

[6] L. J. Colfer and C. Y. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," *IRPN: Innovation & Organizational Behavior (Topic)*, 2016 (⇒ 238).

[7] E. Leo, "Breaking mirror for the customers: The demand-side contingencies of the mirroring hypothesis," *Cont. Man. Res.*, vol. 18, pp. 35–65, Mar. 2022 (⇒ 238).

[8] M. Dorner, M. Capraro, O. Treidler, *et al.*, *Taxing collaborative software engineering*, 2023. arXiv: 2304.06539 [cs.SE] (⇒ 238).

[9] M. Choudaray and M. Cheng, "Export Control," in *Open Source Law, Policy and Practice*, Oxford University Press, Oct. 2022, ISBN: 9780198862345. eprint: https://academic.oup.com/book/0/chapter/378967490/chapter-pdf/49854057/oso-9780198862345-chapter-12.pdf (⇒ 238).

[10] A. Pannier, *Software power: The economic and geopolitical implications of open source software*, 2022 (⇒ 238).

[11] P. M. Institute, *A guide to the Project Management Body of Knowledge (PMBOK guide)*, 7th. Newton Square, PA: PMI, 2021, ISBN: 9781628256673 (⇒ 238).

[12] J. Varajão, R. P. Marques, and A. Trigo, "Project management processes – impact on the success of information systems projects," *Informatica*, vol. 33, no. 2, pp. 421–436, 2022, ISSN: 0868-4952 (⇒ 238).

[13] S. Pretorius, H. Steyn, and T. Bond-Barnard, "The relationship between project management maturity and project success," *J. Mod. Proj.*, vol. 10, pp. 219–231, Mar. 2023 (⇒ 238).

[14] S. Moradi, K. Kähkönen, and K. Aaltonen, "From past to present- the development of project success research," *J. Mod. Proj.*, vol. 8, no. 1, Apr. 2022 (⇒ 238).

[15] P. M. Institute, *A guide to the Project Management Body of Knowledge (PMBOK guide)*, 6th. Newton Square, PA: PMI, 2017, ISBN: 9781628251845 (⇒ 238).

[16] W. E. Deming, *Out of the Crisis* (MIT Press Books). The MIT Press, Dec. 2000, vol. 1, ISBN: 9780262541152 (⇒ 238).

[17] R. Hayes, "Why japanese factories work," *HBR*, vol. 59, pp. 56–66, Jan. 1981 (⇒ 238).

[18] S. Spear and H. Bowen, "Decoding the dna of the toyota production system," *HBR*, vol. 77, Sep. 1999 (⇒ 238).

[19] R. Bohn, "Stop fighting fires," *HBR*, vol. 78, pp. 83–91, Jul. 2000 (⇒ 238).

[20] D. L. Parnas, "Structured programming: A minor part of software engineering," *Information Processing Letters*, vol. 88, no. 1, pp. 53–58, 2003, To honour Professor W.M. Turski's Contribution to Computing Science on the Occasion of his 65th Birthday, ISSN: 0020-0190 (⇒ 238).

[21] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Physical Review E*, vol. 68, no. 4, Oct. 2003 (⇒ 238).

[22] D. Hyland-Wood, D. Carrington, and S. Kaplan, "Scale-free nature of java software package, class and method collaboration graphs," in *Proceedings of the 5th International Symposium on Empirical Software Engineering*, 2006 (⇒ 238).

[23] L. Šubelj and M. Bajec, "Software systems through complex networks science: Review, analysis and applications," in *Proceedings of the First International Workshop on Software Mining*, ser. SoftwareMining '12, Beijing, China: ACM, 2012, pp. 9–16, ISBN: 9781450315609 (⇒ 238).

[24] K. Szabados, "Structural analysis of large ttcn-3 projects," in *Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, ser. TESTCOM '09/FATES '09, Eindhoven, The Netherlands: Springer-Verlag, 2009, pp. 241–246, ISBN: 9783642050305 (⇒ 238).

[25] K. Szabados, "Quality aspects of ttcn-3 based test systems," Ph.D. dissertation, Eötvös Loránd University, Nov. 2017 (⇒ 238).

[26] A. Moura, Y. Lai, and A. Motter, "Signatures of small-world and scale-free properties in large computer programs," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 68 1 Pt 2, p. 017 102, 2003 (⇒ 238).

[27] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in oo programs," *Commun. ACM*, vol. 48, no. 5, pp. 99–103, May 2005, ISSN: 0001-0782 (⇒ 238).

[28] N. LaBelle and E. Wallingford, "Inter-package dependency networks in open-source software," *CoRR*, vol. cs.SE/0411096, 2004 (⇒ 238).

[29] G. Kohring, "Complex dependencies in large software systems," *Advances in Complex Systems*, vol. 12, Nov. 2011 (⇒ 238).

[30] C. Taube-Schock, R. J. Walker, and I. H. Witten, "Can we avoid high coupling?" In *Proceedings of the 25th European Conference on Object-Oriented Programming*, ser. ECOOP'11, Lancaster, UK: Springer-Verlag, 2011, pp. 204–228, ISBN: 9783642226540 (⇒ 239).

[31] M. A. Mamun, C. Berger, and J. Hansson, "Effects of measurements on correlations of software code metrics," *Empir. Softw. Eng.*, vol. 24, Aug. 2019 (⇒ 239).

[32] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Towards a theoretical model for software growth," in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, 2007, pp. 21–21 (⇒ 239).

[33] C. P. Smith, "A software science analysis of programming size," in *Proceedings of the ACM 1980 Annual Conference*, ser. ACM '80, New York, NY, USA: ACM, 1980, pp. 179–185, ISBN: 0897910281 (⇒ 239).

[34] H. Zhang and H. B. K. Tan, "An empirical study of class sizes for large java systems," in *14th Asia-Pacific Software Engineering Conference (APSEC'07)*, 2007, pp. 230–237 (⇒ 239).

[35] H. Zhang, "Exploring regularity in source code: Software science and zipf's law," in *2008 15th Working Conference on Reverse Engineering*, 2008, pp. 101–110 (⇒ 239).

[36] H. Zhang, H. B. K. Tan, and M. Marchesi, "The distribution of program sizes and its implications: An eclipse case study," *CoRR*, vol. abs/0905.2288, 2009. arXiv: 0905.2288 (⇒ 239).

[37] I. Herraiz, D. Germán, and A. E. Hassan, "On the distribution of source code file sizes," in *ICSOFT 2011 - International Conference on Software and Data Technologies*, vol. 2, Jan. 2011, pp. 5–14 (⇒ 239).

[38] R. Tonelli, G. A. Pierro, M. Ortu, and G. Destefanis, "Smart contracts software metrics: A first study," *PLoS ONE*, vol. 18, Jan. 2023 (⇒ 239).

[39] N. Bartha, "Scalability on it projects," M.S. thesis, Eötvös Loránd University, 2016 (⇒ 239).

[40] L. Hatton, "Conservation of information: Software's hidden clockwork?" *IEEE Trans. Softw. Eng.*, vol. 40, no. 5, pp. 450–460, 2014 (⇒ 239).

[41] L. Hatton and G. Warr, "Strong evidence of an information-theoretical conservation principle linking all discrete systems," *Royal Society Open Science*, vol. 6, p. 191 101, Oct. 2019 (⇒ 239).

[42] M. Lehman and J. Fernandez-Ramil, "Rules and tools for software evolution planning and management," *ASE*, vol. 11, pp. 15–44, Jan. 2001 (⇒ 239).

[43] M. M. Lehman and J. F. Ramil, "Evolution in software and related areas," in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, ser. IWPSE '01, Vienna, Austria: ACM, 2001, pp. 1–16, ISBN: 1581135084 (⇒ 239).

[44] M. Lehman, D. Perry, and J. Ramil, "On evidence supporting the feast hypothesis and the laws of software evolution," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, 1998, pp. 84–88 (⇒ 239, 240).

[45] M. J. Lawrence, "An examination of evolution dynamics," in *Proceedings of the 6th International Conference on Software Engineering*, ser. ICSE '82, Tokyo, Japan: IEEE CS Press, 1982, pp. 188–196 (⇒ 239).

[46] C. Izurieta and J. Bieman, "The evolution of freebsd and linux," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ser. ISESE '06, Rio de Janeiro, Brazil: ACM, 2006, pp. 204–211, ISBN: 1595932186 (⇒ 239, 240).

[47] C. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 493–509, 1999 (⇒ 239, 240).

[48] A. Israeli and D. Feitelson, "The linux kernel as a case study in software evolution," *Journal of Systems and Software*, vol. 83, pp. 485–501, Mar. 2010 (⇒ 239, 240).

[49] K. Johari and A. Kaur, "Effect of software evolution on software metrics: An open source case study," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 1–8, Sep. 2011, ISSN: 0163-5948 (⇒ 239, 240).

[50] R. Potvin and J. Levenberg, "Why google stores billions of lines of code in a single repository," *Commun. ACM*, vol. 59, no. 7, pp. 78–87, Jun. 2016, ISSN: 0001-0782 (⇒ 239).

[51] W. Turski, "The reference model for smooth growth of software systems revisited," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 814–815, 2002 (⇒ 239).

[52] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens, "What does it take to develop a million lines of open source code?" In *Open Source Ecosystems: Diverse Communities Interacting*, vol. 299, Jun. 2009, pp. 170–184, ISBN: 978-3-642-02031-5 (⇒ 239, 240).

[53] J. Waveren, "The quake iii arena bot," Jan. 2001 (⇒ 244).

[54] K. Szabados, "Parallelising semantic checking in an ide: A way toward improving profits and sustainability, while maintaining high-quality software development," *Acta Universitatis Sapientiae, Informatica*, vol. 15, no. 2, pp. 239–266, 2023 (⇒ 249).

Received: 06.09.2024; *Revised: 09.12.2024*; *Accepted: 10.12.2024*