Maintainer Retirement under Reversible Whitespace Changes

Ansel M. Calderwick
Software Stewardship Laboratory
Greyfen Institute for Software Systems
Norwich
United Kingdom

capstract:2611.00065v1 [cs.SE] 20 Nov 2026

***Abstract*—We study a retirement rule that permits an incumbent software maintainer to leave once a designated successor holds a majority of the project's line credit. We replay six successive appointments in ten released Python packages. In alternate terms an incoming maintainer inserts one trailing space per eligible line or removes the space inserted in the previous term. The transformation preserves the parsed program. Each incoming maintainer acquires credit for 87,490 of 111,377 nonblank lines across the corpus, enough to qualify the outgoing roster in every package. The three removal terms restore the exact release tree and assign the edited lines to a new author each time. Processing the largest files first reduces the median number of file commits needed to qualify from 15 to 6.5. Whitespace-insensitive blame yields no eligible retirements. Under ordinary line attribution, the same released source can support successive retirements without any additional code entering the project.**

***Index Terms*—software maintenance, maintainer succession, code ownership, repository mining, Git blame**

I. Introduction

The runtime package at release 23.2.0 of attrs contains 4,443 nonblank lines. One file supplies 1,912 of those lines when a new maintainer inserts a trailing space on each eligible line. With the appropriate second file, the new maintainer holds a majority of line credit. An incoming maintainer can obtain the same credit a second time by removing those spaces.

Repository-based assessments use code ownership as evidence of maintenance capacity [1], [2]. We use that evidence for a retirement decision. An outgoing roster qualifies to retire once its designated successor holds more than half of the current line credit. On handover the successor takes office and becomes eligible through a fresh successor under the same rule. There is no minimum elapsed service period. The condition for an orderly departure is the availability of an appropriately credited replacement.

We evaluate this case with a reversible intervention. Six successive maintainers take turns adding and removing trailing spaces in ten Python packages. Each maintainer is given a new Git author identity. In every second term we restore the release tree exactly. We measure whether the roster can retire at each handover, how many file commits it takes to qualify, and which earlier authors are visible to the auditor after the source is restored.

II. Related Work

Avelino et al. [3] describe the takeover of open-source projects after their original core developers leave. Their study identifies work that occurs independently of version history, including establishing contact and obtaining access. Bus-factor methods estimate where knowledge is concentrated while the current maintainers are still available [1], [2], [4]. An orderly departure presents a different problem. The continuity decision concerns coverage held by a particular incoming roster. Counting how many other authors might leave does not establish that the successor is among the remaining holders.

Git line attribution is used in studies of how source code is co-edited [5]. Git can compare versions with whitespace ignored [6]. A content-based audit should also be able to recognize an exact return to a previous tree. This is a stricter condition than whitespace equivalence. The experiment uses it to separate the effect of changing programmers from the effect of changing programs.

III. Method

A. Release snapshots

We used the same ten snapshots for every condition (Table I). This is a convenience sample of tagged Python packages spanning command-line tools, web frameworks, testing, packaging, and serialization. Four are Pallets projects. We fetched complete commit ancestry to each tag and restricted the audit to tracked Python files in runtime package directories. Separate test and documentation trees are outside scope. We count nonblank physical lines, including comment lines and docstrings.

TABLE I
AUDITED RELEASE SNAPSHOTS. COUNTS INCLUDE ALL TRACKED PYTHON FILES IN THE LISTED RUNTIME ROOTS AND ALL NONBLANK PHYSICAL LINES IN THOSE FILES.

Project	Tag	Commit	Runtime roots	Files	Lines
attrs	23.2.0	9e443b18527d	`src/attr`, `src/attrs`	19	4,443
click	8.1.7	874ca2bc1c30	`src/click`	16	8,201
flask	3.0.0	735a4701d6d5	`src/flask`	24	7,065
jinja	3.1.4	dd4a8b5466d8	`src/jinja2`	25	11,598
marshmallow	3.21.3	b9646e326c51	`src/marshmallow`	13	4,196
packaging	24.1	85442b8032cb	`src/packaging`	14	3,790
pytest	8.2.2	329d3712146e	`src/_pytest`, `src/pytest`	69	27,709
requests	v2.32.3	0e322af87745	`src/requests`	18	4,560
rich	v13.7.1	7f580bdcf07a	`rich`	78	23,244
werkzeug	3.0.3	f9995e967979	`src/werkzeug`	52	16,571
Total				328	111,377

B. Eligibility and term boundaries

We identify a line's author by the lowercased email returned by ordinary `git blame --line-porcelain`. Git returned no email on 44 nonblank lines at release. We include them in the denominator and assign their credit to no roster. Let L be the number of audited lines and $A_s(j)$ the number credited to designated successor s after the jth file commit in a term. That successor qualifies the former roster to retire at

$$j_s(q) = \min\{j : A_s(j) > qL\}, \tag{1}$$

provided the successor reaches the cutoff by the end of the term. We use a strict majority ($q = 0.50$), with 67% and 75% as sensitivity cutoffs. The full term includes every planned file commit even when eligibility occurs earlier. The appointment changes once per term. We record the cutoff crossing in the audit and leave the source schedule fixed.

The original roster consists of all identified authors in the release audit. It is followed by six distinct one-person rosters. When a roster retires, its credit can remain visible in the repository. That credit does not count toward any subsequent roster's eligibility. A roster qualifies its predecessor only through its own attributed lines. Thus the same lines can qualify more than one outgoing roster, but only after credit has passed to another successor.

C. Reversible intervention

We generate the intervention from the release snapshot using CPython 3.14.6. Insertions add one ASCII space before the line ending on eligible nonblank lines. We protect lines intersecting a multiline token and lines containing a type comment. We also protect lines ending in a backslash, where inserting a space can invalidate explicit continuation [7]. Every edited file must have the same parsed syntax tree including type comments, with source positions omitted. None failed this check. An inverse edit restores each original blob from the release tree. The inverse does not strip whitespace that was already in the release.

We use Git 2.43.0 to write one commit per changed file and one author per term. Odd terms apply the insertions. Even terms apply the inverses. This yields three round trips per project. Each round trip ends in the original whole-repository tree, including paths outside the audited roots. We verify the tree ID at each endpoint and compare every scored line's author email with its release value. All commits are local and have a fixed committer and timestamp. Time to eligibility is measured in commits because the intervention does not use elapsed time as an input.

D. Schedules and controls

The lexical schedule processes files in ascending path order. The largest-first schedule places files in descending order by the number of changed lines, breaking ties by path. Both schedules edit exactly the same files in each term. We score the affected path after each commit, when the other paths have not changed in that step, and verify the accumulated count with a full audit at the end of the term. The two schedules provide a comparison of the amount of work needed for retirement before the term ends. If each file transfers its observed changed lines, largest first also gives the smallest possible number of files for any fixed cutoff. The sum of the j largest files is at least the sum of any other j files.

We run two controls. First, we rerun all endpoint audits with `git blame -w` and compare those authors with a whitespace-insensitive audit at release. This tests a retirement procedure in which whitespace edits are excluded from ownership accounting. Second, each of the six successor identities writes a metadata-only commit in a separate replay from the same release. We leave the tree unchanged at every commit. This tests whether an announcement of the successor in version history is sufficient to transfer credit.

IV. RESULTS

A. The transfer of retirement eligibility

The audited releases contain 111,377 nonblank lines in 328 files. We edit 87,490 lines in 325 files per term. The smallest transfer is 62.4% of audited lines in Flask. The largest is 87.0% in Rich. In ordinary blame every edited line goes to the current successor in both directions. Each original roster and each of the first five successors therefore qualifies for retire-

ment. At the majority cutoff there are six eligible handovers per package, or 60 across the ten packages. The sixth successor has no designated replacement within the experiment.

The insertion and removal passes give the same eligibility counts for a fixed schedule. In particular, the maintainer appointed after an insertion term receives the same credit for removing each space as the outgoing maintainer received for adding it. The relevant comparison is between consecutive file versions. At the end of the second term the working tree is the release tree, while all 87,490 edited lines are credited to the second successor. Every subsequent round trip gives the same result with a fresh name.

Fig. 1 reports the two comparisons with the release. There is no content difference at the end of terms 2, 4, and 6. The share of changed author records is the same at all six endpoints. If a maintenance office audits two successive release trees without their histories, it cannot observe those retirements from the trees alone.

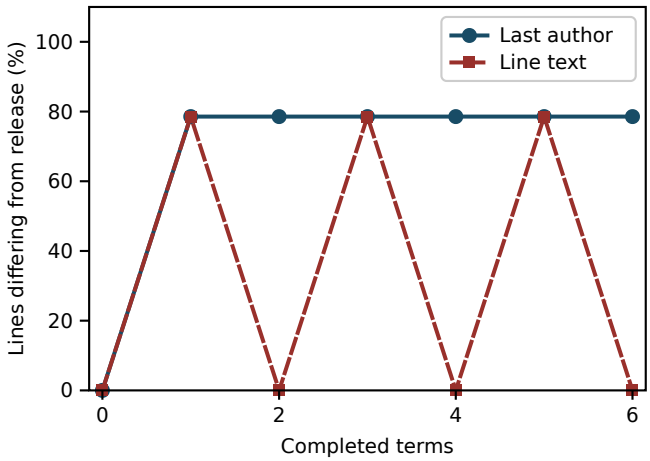


Fig. 1. Pooled share of audited lines differing from the release by text or by last author. Term 0 is the release. Even-numbered terms restore the release tree.

B. File commits required for eligibility

Table II gives the first qualifying commit under each file order. At the majority cutoff the median falls from 15 file commits in lexical order to 6.5 in largest-first order. The largest-first order reached the cutoff earlier in every package. In attrs, two commits suffice. The first transfers 1,912 lines from `src/attr/_make.py`. The second raises the total to 2,233 lines, 11 more than required.

TABLE II
FIRST QUALIFYING FILE COMMIT AT THE 50% CUTOFF. EDITED IS THE NUMBER OF FILE COMMITS IN A FULL TERM. CREDIT IS THE SUCCESSOR'S SHARE AT THAT TERM'S END.

Project	Edited	Lexical	Largest first	Credit (%)
attrs	19	6	2	71.1
click	16	11	6	75.2
flask	24	17	11	62.4
jinja	25	15	7	78.8
marshmallow	13	9	2	74.4
packaging	14	11	5	75.7
pytest	68	36	15	84.1
requests	18	15	4	76.9
rich	78	49	11	87.0
werkzeug	50	39	20	70.0

Fig. 2 shows the paths to eligibility across releases of different sizes. We report progress as the proportion of all files scheduled for editing. In Requests the cutoff is reached after 15 of 18 file commits in lexical order. Starting with the largest files moves the decision to the fourth commit.

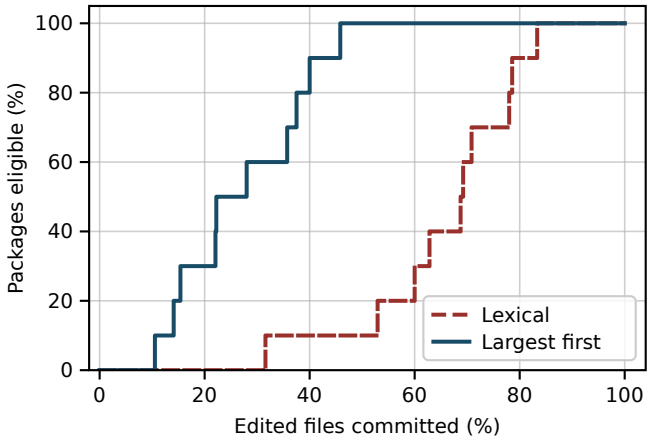


Fig. 2. Cumulative share of packages whose outgoing rosters qualify to retire by a given fraction of the term. The denominator of progress is the number of edited files in that package.

There are fewer eligible handovers at higher cutoffs. At 67% we observe 9 eligible packages. At 75% there are 6. With the latter requirement, Click needs all 16 file commits in either order, although six suffice for a majority under largest first. Reordering files changes when a project qualifies. Reordering cannot make a project eligible if the complete term transfers too few lines.

C. Credit with whitespace ignored

There were zero changes to release attribution under the whitespace-insensitive comparison at any of the 120 endpoints across the two schedules. No successor became eligible under that audit. No author acquired line credit from metadata-only commits either. Together the controls locate the effect in the last-edit attribution of changed file content. A commit that restores the entire release tree transfers credit because the imme-

diately preceding tree is different. A commit that retains the current tree transfers none.

V. Discussion

With the retirement rule used here, a project can authorize another departure whenever it can deliver a new successor across the threshold. The same set of editable lines can serve every term under ordinary blame. At the end of the first round trip in this study, the release has passed through two eligible departures and has the same repository tree as before the first appointment. A third appointment can begin with the first intervention verbatim.

There are two separable decisions for a continuity office. It can make a once-only handover decision for each newly appointed successor. It can also ask whether the organization has gotten better at replacing staff after successive departures. Our measurements distinguish those decisions. Eligibility can recur even when the code audit has found no evidence of additional maintainable source. If the procedure requires changes in demonstrated responsibility, it needs observations that still distinguish roles when file content and author credit move independently. Review activity and other records of participation have been used for this purpose [4].

A simple control for a release-based process is to score both the current tree and the closest prior tree with equal content. Where both occur, a second query with whitespace ignored identifies cases in which the current distribution of authority depends entirely on formatting. These checks address different parts of the decision. The first identifies a return to a previous source state. The second also applies at intermediate endpoints, where the trees differ by spaces.

VI. Threats to Validity

Our ten projects are a convenience sample of Python releases. The eligible line shares depend on the proportion of the files occupied by multiline strings and other protected constructs. Other languages may require other transformations. The experiment includes no claim that typical formatting reviews are this extensive. A layout rewrite that splits or moves lines would also require a different line-matching check.

Git author emails are identity labels. The six experimental appointments were simulated, and real contributors were never removed from a project. We measure the behavior of a rule applied to Git records. We do not observe either a developer’s readiness to retire or a successor’s ability to take over independent work. The majority convention weights every line equally. A procedure based on responsibility for specific modules or release duties can give a different decision. AST equality controls changes to the parsed program in one Python version. Other tooling can respond to whitespace in source text.

VII. Conclusion

A maintenance rule that treats last-edit credit as evidence of completed succession can release several successive rosters against one release tree. Reversing a whitespace change restores the file content and gives the next maintainer credit for the same lines. A retirement approval therefore depends on the direction from which a maintainer arrived at the tree, even when the project could distribute exactly the same source both before and after the appointment.

Appendix A. Replication Details

Use the tag and commit abbreviation in Table I to locate each release. Enumerate package paths in lexical order. Keep both the original Git blob and the transformed blob for each edited file. For each term write the appropriate blobs to an index initialized from the preceding term’s endpoint. After each file, record the index tree and write a commit with the current term’s author and previous commit as its parent. Count the term author’s nonblank records in `git blame --line-porcelain`, summing across files. The endpoint following each removal term must have the same tree ID as the release. Run `git blame -w` at that point and at release for the whitespace control. Section III specifies token protection, counting, and file order.

References

[1] G. Avelino, L. Passos, A. Hora, and M. T. Valente, “A Novel Approach for Estimating Truck Factors,” in *Proc. 2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 2016, pp. 1–10.

[2] M. Ferreira, T. Mombach, M. T. Valente, and K. Ferreira, “Algorithms for Estimating Truck Factors: A Comparative Study,” *Software Quality Journal*, vol. 27, no. 4, pp. 1583–1617, 2019, doi: 10.1007/s11219-019-09457-2.

[3] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, “On the Abandonment and Survival of Open Source Projects: An Empirical Investigation,” in *Proc. 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–11.

[4] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, and V. Kovalenko, “Bus Factor in Practice,” in *Proc. 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2022, pp. 97–106.

[5] C. Gote, I. Scholtes, and F. Schweitzer, “Analysing Time-Stamped Co-Editing Networks in Software Development Teams Using git2net,” *Empirical Software Engineering*, vol. 26, pp. 75, 2021, doi: 10.1007/s10664-020-09928-2.

[6] Git Project, “git-blame Documentation,” 2023.

[7] Python Software Foundation, “Lexical Analysis,” 2026.