Standing on the Shoulders of Decaying Giants

Anonymous Author(s)

***Abstract*—Software engineering has improved reproducibility through replication packages that let others rerun tools and reproduce results. A replication package, however, is a snapshot, and its value as a baseline can diminish in three ways: the tool may stop working as intended, its approach may fit today's world worse, or the evaluation setup may make comparing against it unsound. We unify these facets as *tool decay*. Left unchecked, tool decay means that outperforming an older tool need not demonstrate genuine progress. We study tool decay in merge conflict resolution, a long-established research area, as a first case study. We combine two complementary lines of evidence: (i) A citation analysis shows that comparisons increasingly copy published baseline numbers instead of rerunning tools, so decay goes unmeasured, and that chains of superiority claims often break. (ii) Two experiments demonstrate decay in five of eight merge tools and show that a simple method of undoing decay overshoots. Together, these findings suggest an erosion of comparability across studies: a systemic threat to the progress of software engineering research. To counter tool decay, we call for establishing shared practices to anticipate and document decay, to undo it systematically, and to sustain fair comparisons.**

***Index Terms*—tool decay, software aging, replication crisis, empirical software engineering, merge tools**

I. Introduction

Over the past two decades, research communities, first in biomedicine and psychology, have recognized that a large share of published findings does not hold up when studies are repeated [1]. In a survey across disciplines, more than 70% of researchers reported having failed to reproduce prior experiments [2]. The main remedies are rigorous study design and statistical analysis, combined with openly archiving data, materials, and code [3]. Software engineering (SE) research has followed suit: replication packages, which often contain a prototype implementation of the proposed approach (i.e., a tool), are becoming the norm, encouraged by artifact evaluation tracks and badges [4], [5]. This enables others to rerun the study and obtain the reported results.

Tools, however, are not only rerun but also serve as baselines: a new tool $\mathcal{N}$ claims to outperform the state of the art $\mathcal{O}$, which in turn outperformed its predecessors. This raises a question replication packages leave open: how can an older approach be evaluated fairly today? An approach can be evaluated only as a tool, and the replication package supplies one: $\mathcal{O}$, its authors' own, frozen at release while the world moves on. We call the resulting loss of soundness of evaluations against $\mathcal{O}$ *tool decay*. Its three parts (Fig. 1) set in independently, but their remedies stack, each presupposing the previous one.

Artifact decay affects the implementation and execution of $\mathcal{O}$, e.g., through changes to the runtime, broken dependencies, or retired services. Its remedy is *repair*. *Field decay* hits $\mathcal{O}$'s

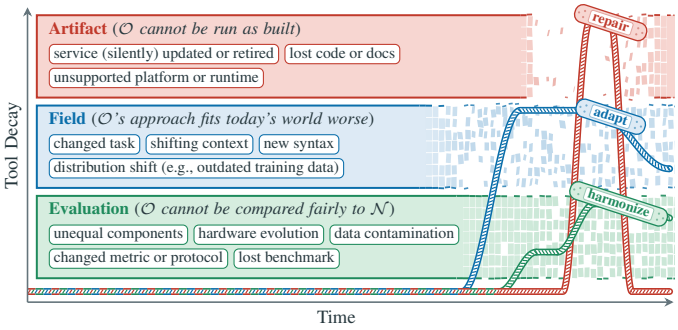


Fig. 1. Overview of tool decay and example facets. Artifact, field, and evaluation decay vary independently in onset (possibly at release), speed, and extent. A remedy (repair, adapt, harmonize) undoes each to a varying extent.

approach in today's world. On its original inputs, $\mathcal{O}$ may reproduce its paper's results, but answers a question about the past that a newer tool need not answer. On today's inputs, $\mathcal{O}$ fails slightly to drastically, even where a tiny adaptation, e.g., one grammar rule for new syntax, would suffice. The remedy is *minimal adaptation*, minimal being a judgment clear only at the extremes. *Evaluation decay* makes the evaluation unsound for one tool (e.g., data contamination) or between tools (e.g., unequal hardware, metrics, or inputs). Its remedy is a *harmonized* setup. Each remedy must fairly undo decay, i.e., without turning $\mathcal{O}$ into another tool or overshooting and boosting it into $\mathcal{O}^{+}$.

Unremedied, decay has two consequences that hinder scientific progress. First, a rerun meets the decayed tool $\mathcal{O}^{-}$, so $\mathcal{N}$ may beat its state of decay rather than its approach. Only comparing $\mathcal{N}$ under harmonized conditions with $\mathcal{O}^{*}$, i.e., $\mathcal{O}$ repaired and minimally adapted, would *isolate* $\mathcal{N}$'s conceptual advantage. Second, wins stop being *transitive*, though the field relies on transitivity when it copies published numbers instead of rerunning a baseline or skips a tool a newer one has beaten (Sec. II-B). A chain of wins holds only if every link compares working tools on the same question and under a harmonized setup. Otherwise, individual wins do not guarantee cumulative progress [6], leaving it unclear whether the field is advancing or stagnating. Maintainers rarely undo decay, so the comparing researcher must address it ad hoc, paper by paper.

Tool decay exists on top of deficiencies in replication packages [7] and mistakes in comparing papers. It includes software aging through "lack of movement" [8], consistent with Lehman's law of continuing change [9]. Even a tool that still runs and reproduces its published results is not necessarily a sound baseline for today's problems. Many studies already address individual facets of tool decay, e.g., by rerunning his-

torical SAT solvers on equal hardware and benchmarks [10], but a unified account of these facets and their remedies is still missing. We hypothesize that, fairly and minimally adapted, older approaches often keep up with their successors. Yet the prevalence of tool decay and its implications for claims of progress remain largely unknown.

To ground our concept in empirical evidence, we take merge conflict resolution, a well-established SE subdomain with more than a decade of tool comparisons, as a case study. There, even GIT, the universal baseline of merge tools, decays: it replaced its default merge algorithm in 2021 and, since 2025, silently runs the new one when the old one is requested [11]. We combine a citation analysis (Sec. II) and two experiments on field and artifact decay (Sec. III). Together, these analyses and experiments motivate the concept and yield three main contributions:

- introducing *tool decay* as a holistic concept that unites known but separately and informally treated threats to measurements and comparisons and explains why wins over older tools need not mean progress;
- a citation analysis showing that maintainers rarely undo decay, that comparisons have shifted from rerunning to copying published numbers, and that chains of wins break, in some cases through decay;
- two experiments demonstrating tool decay in five of eight measured tools, and why undoing it is not clear-cut.

Replication package: doi.org/10.5281/zenodo.23593165.

II. Evidence of Decay in Literature

A. Method

We started from 41 papers that present merge tools, both mergers and conflict resolvers. They stem from our collection of merge conflict resolution literature, which we started with a keyword search and have since extended by repeated backward and forward snowballing. We collected their 448 citing papers from OpenAlex, Semantic Scholar, Google Scholar, and this collection, keeping only peer-reviewed publications and doctoral theses. We count one comparison per ⟨comparing paper, tool⟩ pair, treating papers that describe the same tool, e.g., a conference paper and its journal extension, as one. We labeled the papers qualitatively in two stages, where three independent runs of LLM agents pre-labeled every item at each stage. *Stage A* labeled each citing paper from its abstract as 'mentioning', 'using', or 'comparing' against a tool. *Stage B* labeled each ⟨comparing paper, tool⟩ pair from the full text. It recorded how the comparing paper handled the tool: reran it as is, reran it as is with failures counted, adapted it, reimplemented it, copied numbers, or excluded it. It further recorded practices such as skipping a tool because another one reportedly outperformed it. Separately, to see how tool authors handle decay, we labeled all 237 updates that the authors or maintainers of the 35 audited tools made since the original version (releases, commits, follow-up papers), likewise in three runs, as undoing decay, hardening against it, weakening the tool against later decay, changing results on capabilities the original version had, or introducing decay. Every label is anchored in a verbatim quote from its source (paper or artifact). We manually adjudicated 26% of the candidate comparisons, including all on which the three runs disagreed and all flagged as contradicting the comparing paper's authors or as excluded due to decay. The full protocol and all data are part of our replication package.

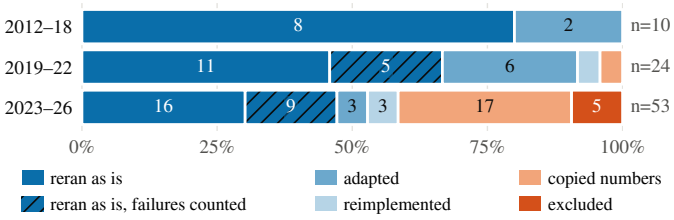


Fig. 2. How comparing papers handled a prior tool, by the comparing paper's publication period. Each comparison is counted once. n = comparisons.

TABLE I
Tool Updates Since the First Tool Release.

Updates	Undoes decay	Introduces decay	Hardens	Weakens	Changes results
237	24	32	71	38	77

B. Results

From rerunning to copying. The analysis yields 87 comparisons in 39 comparing papers. Until 2018, all 10 comparisons reran the baseline. Since 2023 (Fig. 2), 17 of 53 comparisons copied published numbers and five excluded a tool that was unavailable or defective since its release. Copying spares the cost of undoing decay, but carries over outdated numbers.

Decay is rarely documented. Only eight of 87 comparisons name decay of the compared tool, half of them in a single paper [12]. For 11 more, we infer it from the tool's artifact, e.g., an outdated language level. Several tools were already decayed at publication: three were never released, and three lacked their trained models or data. Decay thus seldom appears in the text as a cause but shows indirectly, in tools being copied or (silently) excluded.

Maintainers repair, and their tools drift. Table I counts the 237 updates by their effect on decay (one update can have several). Of them, 24 undo decay, and those stated explicitly mostly fix the build or runtime. Others introduce decay, mostly unannounced. Hardening against decay is more common, but some updates make later decay more likely. 77 updates change results on capabilities the original version had: they may yield $\mathcal{O}^*$, leave $\mathcal{O}^-$, or boost it to $\mathcal{O}^+$, often a judgment call. "The latest version" may thus be any of the three, yet 69 of 87 comparisons do not state the version, 40 of them although several existed.

Broken and questionable chains. Later papers repeat earlier claims of superiority and treat them as transitive, even when subsequent evaluations question or contradict those claims. Of the 11 broken or questionable chains we found, all

but five stem from noise or plain mistakes, e.g., a miscopied number. The other five are possibly due to decay (C1–C5 below; $\succ$: outperforms, $\approx$: no difference).

C1: AUTOMERGE $\overset{[13],\,[14]}{\succ}$ JDIME $\overset{[15],\,[16]}{\approx}$ AUTOMERGE; JDIME skipped [17]

C2: INTELLIMERGE $\overset{[18]}{\succ}$ JFSTMERGE $\overset{[16],\,[19],\,[20]}{\succ}$ INTELLIMERGE; JFSTMERGE skipped [21]

C3: SPORK $\overset{[22],\,[23]}{\succ}$ JDIME; JDIME excluded as unusable, SPORK's win questioned [12]

C4: neural resolvers $\overset{[24]\text{–}[27]}{\succ}$ JSFSTMERGE $\overset{[28]}{\succ}$ neural resolvers

C5: DEEPMERGE was never released [12]; a non-equivalent reimplementation's results [24] were later reported as "DEEPMERGE" results [27]

III. Evidence of Decay in Experiments

A. Experimental Setup

Tools. We run eight merge tools, each in the runnable version closest to its release and on its original Java runtime: JDIME [29], JFSTMERGE [30], INTELLIMERGE [18], SPORK [22], MERGIRAF [31], MASTERY [19], IVN [12], and CHATMERGE [32]. IVN had already decayed, as it runs only with its maintainers' one-line repair, which we applied before the experiment.

Data. We sample 1,700 merges from 410 Java projects on GitHub: 100 per year from 2010 to 2026. We choose highly starred Java projects (400+ stars) plus those the tools' own papers evaluated on. We keep merges in which GIT v2.56 reports a content conflict in a Java file, and take one file per merge (picked at random if several), with all its conflict chunks. No project contributes more than five merges per year. We leave out 32 merges whose resolution is not valid Java (18 with committed conflict markers). Exp. 1 runs seven tools on the remaining 1,668 merges, and Exp. 2 runs CHATMERGE on the 1,115 of them whose file has a single conflict chunk.

Metric. To evaluate a merge's correctness, we follow MASTERY's equivalence check [19]: a merge is *correct* if the tool returns a clean file that is equivalent to the developer's resolution, ignoring comments and formatting differences [16]. In Exp. 1, we compare each tool's correct-merge rate before and after its release date (two-sided Fisher exact test, Holm-adjusted over the seven tools). In Exp. 2, we score three configurations of CHATMERGE on the same merges, per merge year (paired McNemar test over all years).

B. Exp. 1 (field decay): Are merge tools, run as released, less often correct on merges made after their release?

Of the seven tools, six merge correctly less often after their release (Fig. 3), significantly so for four ($p_{\text{Holm}} < 0.04$): JDIME from 14.7% to 8.4%, JFSTMERGE from 27.4% to 21.8%, SPORK from 39.4% to 29.1%, and MASTERY from 38.9% to 30.0%. One cause is new syntax: JDIME, whose parser stops at Java 7, is correct on 14.5% of the merges that need at most Java 7 but on 0.5% of those needing newer syntax. INTELLIMERGE rises slightly, from 7.5% to 8.7% ($p_{\text{Holm}} = 0.42$). We find little distribution shift, i.e., merges

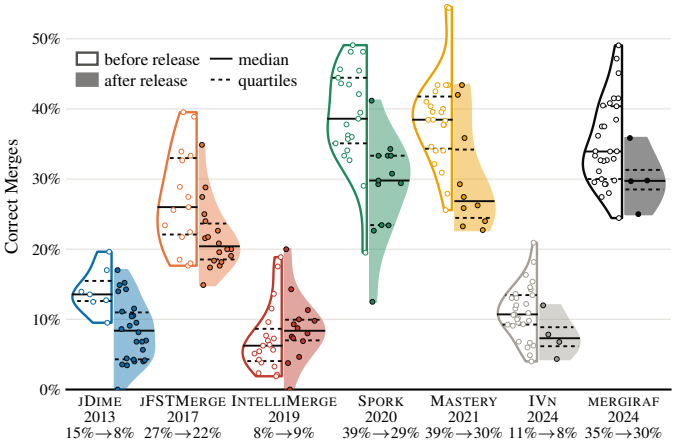


Fig. 3. Correct-merge rate of each tool as released (mean rates before/after release noted below). Each dot represents ~50 merges per six-month period.

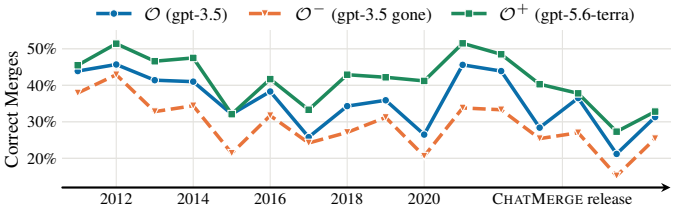


Fig. 4. CHATMERGE correct merges: original $\mathcal{O}$, decayed $\mathcal{O}^-$, boosted $\mathcal{O}^+$.

are not simply becoming harder over time: splitting each tool's pre-release merges at their median date, for six of the seven tools (all but JFSTMERGE) the newer half is merged correctly as often as the older half (pooled -0.7 points, 95% CI $[-2.6, +1.3]$).

C. Exp. 2 (artifact decay): When an LLM component in a merge tool is retired, does a successor LLM repair it?

CHATMERGE resolves a conflict itself when its classifier is confident and otherwise uses gpt-3.5-turbo, which OpenAI retired on 23 October 2026, incidentally the day this paper was due, with gpt-5.6-terra as its official successor [33]. We ran CHATMERGE as released. It used gpt-3.5-turbo on 294 of the 1,115 merges (26.4%). CHATMERGE turns all failed LLM calls into unresolved conflicts, so after retirement these 294 merges fail (simulated, Sec. IV) and its correct-merge rate significantly drops from 37.1% for the original $\mathcal{O}$ to 30.2% for the decayed $\mathcal{O}^-$ ($p < 0.001$, Fig. 4). Swapping in the official successor boosts it to $\mathcal{O}^+$, which significantly outperforms $\mathcal{O}$: its correct-merge rate rises to 42.8% ($p < 0.001$).

IV. Discussion

Interpreting our evidence. Our citation analysis shows that the field increasingly relies on copying published results (trusting they still hold, or because the tool is too decayed to be rerun), excluding decayed tools, or skipping tools that a newer tool has beaten, assuming that wins are transitive. The 11 broken or questionable chains of wins we found (Sec. II-B) challenge that assumption.

Exp. 1 and Exp. 2 demonstrate that tool decay varies across tools, with different facets affecting each. Some of the significant decay we found may seem small, e.g., JFSTMERGE loses 5.6 and JDIME 6.3 points of correct merges, but smaller differences were deemed "important" [12] or "significant" [27], and earned "the highest merge accuracy" [19] or "the best performance" [32].

While Exp. 1 shows the combined effect of several facets of field decay, Exp. 2 isolates one facet of artifact decay, a retired LLM component. In our case, the retirement of a commercial LLM cannot simply be undone: its successor performs better and has a different training cutoff, and thus different data contamination. This matches observations that commercial LLMs change behavior over time [34] and that LLM-based SE techniques fare differently with newer models [35], [36]. Additionally, Exp. 2 cannot be repeated, as gpt-3.5-turbo is no longer accessible.

We showed tool decay only for our case study of merge tools, but we expect its facets to exist in many SE domains, so the following addresses the SE community at large.

What makes a fair comparison? Where $\mathcal{O}^-$ ends and $\mathcal{O}^*$ or $\mathcal{O}^+$ begins is subjective and depends on the field. Still, the extremes should be uncontroversial, e.g., the maintainers' one-line repair of IVN [12] we applied (Sec. III-A). Until communities agree on thresholds, we suggest reporting results and error rates for both $\mathcal{O}^-$ and an ad hoc remedied $\mathcal{O}^?$.

Implications of tool decay. Tool decay affects the wider research community and its output. Researchers comparing against an old tool must choose between undoing its decay, which may fail or overshoot, copying its published numbers, and skipping it [37]. Some authors already maintain their tools, but research communities and venues should give more incentive to keep tools maintained while avoiding the drift toward $\mathcal{O}^+$. The goal is that a field's best tool is free of decay and actually is the transitive winner.

Mitigating tool decay. The community standard of archiving on Zenodo or Figshare under a permanent DOI already prevents the most basic decay facet, a vanished artifact [38]. By design, however, archives preserve exactly what decays. We additionally argue that many facets of tool decay can be undone or prevented only if tools and the services they use are open source: only then can later researchers notice which facets affect a tool, and undo them. To undo artifact decay efficiently, complete xBOMs [39] should become a mandatory part of replication packages. We envision additional novel BOM types that formalize the task and evaluation, e.g., a metric entry `accuracy: as in <DOI>`.

Threats to validity. Because we compare aging tools and draw on prior comparisons, our study is itself particularly exposed to tool decay. First, it limits which tools we can run: tools with missing artifacts or models, or built on the retired GPT-3, are excluded by construction, so our experiments understate decay. Second, it forced us to simulate $\mathcal{O}^-$ in Exp. 2 before gpt-3.5-turbo retired. Since then, $\mathcal{O}$ can no longer be run. Third, even our baseline merge sample is decaying: we sample merges that conflict with GIT, the field's standard merge tool. However, GIT replaced its default merge algorithm in 2021 and, since 2025, silently runs the new one when the old one is requested [11]. Further, decay may stay silent [40]: some tools fall back, e.g., when they fail to parse newer Java code. To avoid judging failure or fallback types subjectively, we score only the *correct-merge rate*. Beyond decay, our citation analysis relies on LLM labels. Three independent runs of Claude Opus 5.5 labeled every item, and we manually adjudicated all labeling disagreements and a sample of the rest. Finally, Exp. 1 could simply reflect merges becoming harder; however, our pre-release split shows no such trend.

V. Related Work

While SE reproducibility progressed via artifact evaluations, badges, and DOIs for artifacts [4], [5], [38], [41], replication packages conserve the behavior of aging tools. By design, they preserve exactly what decays. Future comparisons, however, need a maintained tool.

Software aging [8], [9] is countered by software maintenance, a well-established discipline [42], [43]. We view artifact decay as aging of research software. Adaptive maintenance, i.e., reacting to a changed environment [42], maps onto our repair and minimal adaptation.

However, the task of maintenance is a task for the original developer. In our case, though, it becomes a *third-party* task as the comparing researchers must undo each tool's decay, usually ad hoc, paper by paper. Currently there is no formal and holistic process for dealing with tool decay, let alone all its facets. It thus becomes a threat to the validity of comparisons and to cumulative progress.

VI. Future Plans

Beyond extending our study (e.g., attributing each tool's drop to artifact, field, or evaluation decay; controlled experiments isolating single facets), we outline several research directions for the SE community. First, we call for a comprehensive taxonomy of tool decay that captures both shared and field-specific facets. Building on this, xBOMs for replication packages, including new BOM types, would document decay-prone components and, where possible, suggest adequate forward-looking remedies. In parallel, communities must decide which decay facets to address, and through which standardized practices. Extending this study beyond SE would reveal how other research communities address decay and which remedies transfer. Finally, decay and the cost of undoing it should be predicted rather than measured post hoc, so that preventive measures can be taken before a tool decays.

We see our contribution as a step toward making tool decay a first-class concern in research and call for holistic, systematic processes to address this multifaceted phenomenon.

References

[1] Open Science Collaboration, "Estimating the reproducibility of psychological science," *Science*, vol. 349, no. 6251, p. aac4716, 2015, doi: 10.1126/science.aac4716.

[2] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, no. 7604, pp. 452–454, 2016, doi: 10.1038/533452a.

[3] M. R. Munafò, B. A. Nosek, D. V. M. Bishop, K. S. Button, C. D. Chambers, N. Percie du Sert, U. Simonsohn, E.-J. Wagenmakers, J. J. Ware, and J. P. A. Ioannidis, "A manifesto for reproducible science," *Nature Human Behaviour*, vol. 1, no. 1, 2017, doi: 10.1038/s41562-016-0021.

[4] B. Hermann, S. Winter, and J. Siegmund, "Community expectations for research artifacts and evaluation processes," in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 469–480, doi: 10.1145/3368089.3409767.

[5] C. S. Timperley, L. Herckis, C. Le Goues, and M. Hilton, "Understanding and improving artifact sharing in software engineering research," *Empirical Software Engineering (EMSE)*, vol. 26, no. 4, p. 67, 2021, doi: 10.1007/s10664-021-09973-5.

[6] T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel, "Improvements that don't add up: ad-hoc retrieval results since 1998," in *Proc. ACM Conf. on Information and Knowledge Management (CIKM)*, 2009, pp. 601–610, doi: 10.1145/1645953.1646031.

[7] C. Collberg and T. A. Proebsting, "Repeatability in computer systems research," *Comm. ACM*, vol. 59, no. 3, pp. 62–69, 2016, doi: 10.1145/2812803.

[8] D. L. Parnas, "Software aging," in *Proc. Int'l Conf. on Software Engineering (ICSE)*, 1994, pp. 279–287, doi: 10.1109/ICSE.1994.296790.

[9] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.

[10] A. Biere, M. Fleury, N. Froleyks, and M. J. H. Heule, "The SAT museum," in *Proc. Int'l Workshop on Pragmatics of SAT (PoS)*, ser. CEUR Workshop Proceedings, vol. 3545, 2023, pp. 72–87.

[11] Git Project, "Git documentation: Merge strategies," https://git-scm.com/docs/merge-strategies, 2025, git v2.50.0; accessed 2026-10-05.

[12] B. Schesch, R. Featherman, K. J. Yang, B. Roberts, and M. D. Ernst, "Evaluation of version control merge tools," in *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, 2024, pp. 831–839, doi: 10.1145/3691620.3695075.

[13] F. Zhu, F. He, and Q. Yu, "Enhancing precision of structured merge by proper tree matching," in *Companion Int'l Conf. on Software Engineering (ICSEC)*, 2019, pp. 286–287, doi: 10.1109/icse-companion.2019.00117.

[14] G. Seibt, F. Heck, G. Cavalcanti, P. Borba, and S. Apel, "Leveraging structure in software merge: An empirical study," *IEEE Trans. on Software Engineering (TSE)*, vol. 48, no. 11, pp. 4590–4610, 2022, doi: 10.1109/tse.2021.3123143.

[15] B. Shen and N. Meng, "Conflictbench: A benchmark to evaluate software merge tools," *J. Systems and Software (JSS)*, vol. 214, p. 112084, 2024, doi: 10.1016/j.jss.2024.112084.

[16] A. Mori and M. Hashimoto, "On the correctness of software merge," in *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, 2025, pp. 2338–2349, doi: 10.1109/ase63991.2025.00193.

[17] Q. J. Gu and M. Janota, "A universal textual merge strategy based on tokens for version control systems," *Computing Research Repository (CoRR)*, 2026, doi: 10.48550/ARXIV.2604.13813.

[18] B. Shen, W. Zhang, H. Zhao, G. Liang, Z. Jin, and Q. Wang, "Intellimerge: a refactoring-aware software merging technique," *Proceedings of the ACM on Programming Languages (PACMPL)*, vol. 3, no. OOPSLA, pp. 1–28, 2019, doi: 10.1145/3360596.

[19] F. Zhu, X. Xie, D. Feng, N. Meng, and F. He, "Mastery: Shifted-code-aware structured merging," in *Proc. Int'l Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*, 2022, pp. 70–87, doi: 10.1007/978-3-031-21213-0_5.

[20] ——, "On the methodology of three-way structured merge in version control systems: Top-down, bottom-up, or both," *Journal of Systems Architecture*, vol. 145, p. 103011, 2023, doi: 10.1016/j.sysarc.2023.103011.

[21] M. Ellis, S. Nadi, and D. Dig, "Operation-based refactoring-aware merging: An empirical evaluation," *IEEE Trans. on Software Engineering (TSE)*, vol. 49, no. 4, pp. 2698–2721, 2023, doi: 10.1109/tse.2022.3228851.

[22] S. Larsen, J.-R. Falleri, B. Baudry, and M. Monperrus, "Spork: Structured merge for java with formatting preservation," *IEEE Trans. on Software Engineering (TSE)*, vol. 49, no. 1, pp. 64–83, 2023, doi: 10.1109/tse.2022.3143766.

[23] G. Cavalcanti, P. Borba, L. d. Anjos, and J. Clementino, "Semistructured merge with language-specific syntactic separators," in *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, 2024, pp. 1032–1043, doi: 10.1145/3691620.3695483.

[24] A. Svyatkovskiy, S. Fakhoury, N. Ghorbani, T. Mytkowicz, E. Dinella, C. Bird, J. Jang, N. Sundaresan, and S. K. Lahiri, "Program merge conflict resolution via neural transformers," in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*, 2022, pp. 822–833, doi: 10.1145/3540250.3549163.

[25] E. Dinella, T. Mytkowicz, A. Svyatkovskiy, C. Bird, M. Naik, and S. Lahiri, "Deepmerge: Learning to merge programs," *IEEE Trans. on Software Engineering (TSE)*, vol. 49, no. 4, pp. 1599–1614, 2023, doi: 10.1109/tse.2022.3183955.

[26] J. Dong, Q. Zhu, Z. Sun, Y. Lou, and D. Hao, "Merge conflict resolution: Classification or generation?" in *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, 2023, pp. 1652–1663, doi: 10.1109/ase56229.2023.00155.

[27] C. Peng, Z. Zhang, S. Tyszberowicz, Z. Liu, and B. Liu, "From generation to reasoning: Chain-of-thought guided merge conflict resolution," in *Proc. Int'l Conf. on Program Comprehension (ICPC)*, 2026, pp. 331–342, doi: 10.1145/3794763.3794821.

[28] M. Fan, W. Zhang, H. Zhao, and Z. Jin, "Reliable version merging based on deep semantic and logical understanding of critical context," in *Proc. Int'l Symposium on Software Reliability Engineering (ISSRE)*, 2025, pp. 430–439, doi: 10.1109/issre66568.2025.00050.

[29] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: balancing precision and performance," in *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, 2012, pp. 120–129, doi: 10.1145/2351676.2351694.

[30] G. Cavalcanti, P. Borba, and P. Accioly, "Evaluating and improving semistructured merge," *Proceedings of the ACM on Programming Languages (PACMPL)*, no. OOPSLA, pp. 1–27, 2017, doi: 10.1145/3133883.

[31] A. Delpeuch and Mergiraf contributors, "Mergiraf: A syntax-aware git merge driver," https://codeberg.org/mergiraf/mergiraf, 2024, version 0.1.0, accessed 2026-10-21.

[32] C. Shen, W. Yang, M. Pan, and Y. Zhou, "Git merge conflict resolution leveraging strategy classification and llm," in *Proc. IEEE Int'l Conf. on Software Quality, Reliability, and Security (QRS)*, 2023, pp. 228–239, doi: 10.1109/qrs60937.2023.00031.

[33] OpenAI, "OpenAI API documentation: Deprecations," https://web.archive.org/web/20261020162442/https://developers.openai.com/api/docs/deprecations, 2026, shutdown of gpt-3.5-turbo on 2026-10-23; archived 2026-10-20.

[34] L. Chen, M. Zaharia, and J. Zou, "How is ChatGPT's behavior changing over time?" *Harvard Data Science Review*, vol. 6, no. 2, 2024, doi: 10.1162/99608f92.5317da47.

[35] M. Konstantinou, R. Degiovanni, and M. Papadakis, "How well LLM-based test generation techniques perform with newer LLM versions?" arXiv preprint arXiv:2601.09695, 2026.

[36] N. Salsabil, J. Saha, S. B. Dristi, N. Phair, N. J. Mozumder, M. B. Dwyer, and S. Elbaum, "What survives the next model? benchmarking LLM-based techniques against single-prompts," arXiv preprint arXiv:2609.00468, 2026.

[37] A. Muttakin, S. Mondal, and C. K. Roy, "The state of open science in software engineering research: A case study of ICSE artifacts," in *Proc. Int'l Conf. on Software Engineering (ICSE)*, 2026, pp. 2299–2310, doi: 10.1145/3744916.3787813.

[38] D. Beyer and A. Hartmanns, "Reproducibility and replication of research results," *Int'l J. Software Tools for Technology Transfer (STTT)*, vol. 27, no. 4, pp. 397–401, 2025, doi: 10.1007/S10009-025-00816-Y.

[39] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk, "BOMs away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems," in *Proc. Int'l Conf. on Software Engineering (ICSE)*, 2024, pp. 1–13, doi: 10.1145/3597503.3623347.

[40] E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis, "The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements," *PLoS ONE*, vol. 7, no. 6, p. e38234, 2012, doi: 10.1371/journal.pone.0038234.

[41] M. Böhme, "Software engineering reproducibility manifesto," https://mboehme.github.io/manifesto, 2019.

[42] E. B. Swanson, "The dimensions of maintenance," in *Proc. Int'l Conf. on Software Engineering (ICSE)*, 1976, pp. 492–497.

[43] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990, doi: 10.1109/52.43044.