Coding on Copilot

2023 Data Shows Downward Pressure on Code Quality

150m lines of analyzed code + projections for 2024

William Harding, Lead Researcher & CEO
Alloy.dev Research

Matthew Kloster, CTO
GitClear

Published January 16, 2024

Abstract

2023 marked the coming out party for GitHub Copilot. In less than two years' time, the AI programming assistant shot from "prototype" to "cornerstone," used by *millions* of developers across *hundreds of thousands* of businesses [1]. Its unprecedented growth defines a new era in "how code gets written."

GitHub has published several pieces of research on the growth and impact of AI on software development. Among their findings is that developers write code "55% faster" when using Copilot. This profusion of LLM-generated code begs the question: how does the **code quality** and **maintainability** compare to what would have been written by a human? Is it more similar to the careful, refined contributions of a Senior Developer, or more akin to the disjointed work of a short-term contractor?

To investigate, GitClear collected 153 million changed lines of code, authored between January 2020 and December 2023 [A1]. This is the largest known database of highly structured code change data that has been used to evaluate code quality differences [A2].

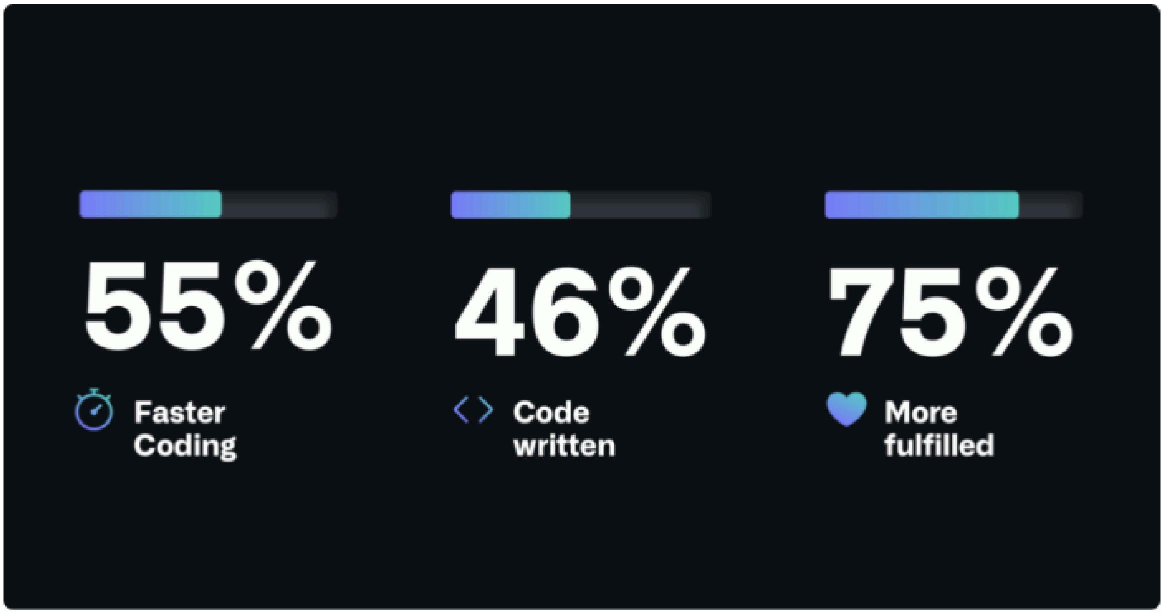
We find disconcerting trends for maintainability. Code churn -- the percentage of lines that are reverted or updated less than two weeks after being authored -- is projected to double in 2024 compared to its 2021, pre-AI baseline. We further find that the percentage of "added code" and "copy/pasted code" is increasing in proportion to "updated," "deleted," and "moved" code. In this regard, code generated during 2023 more resembles an itinerant contributor, prone to violate the DRY-ness of the repos visited.

We conclude with suggestions for managers seeking to maintain high code quality in spite of the momentum opposing that goal.

Table of Contents

GitHub: "55% faster coding. 46% more code written. $1.5 trillion added to GDP"

With numbers like these, little wonder that GitHub's own CEO, Thomas Dohmke, has been taking time from his usual CEO duties to write about the AI revolution. A blog post and research paper he published on GitHub in 2023 tell a heady story about the rapid proliferation of Copilot.



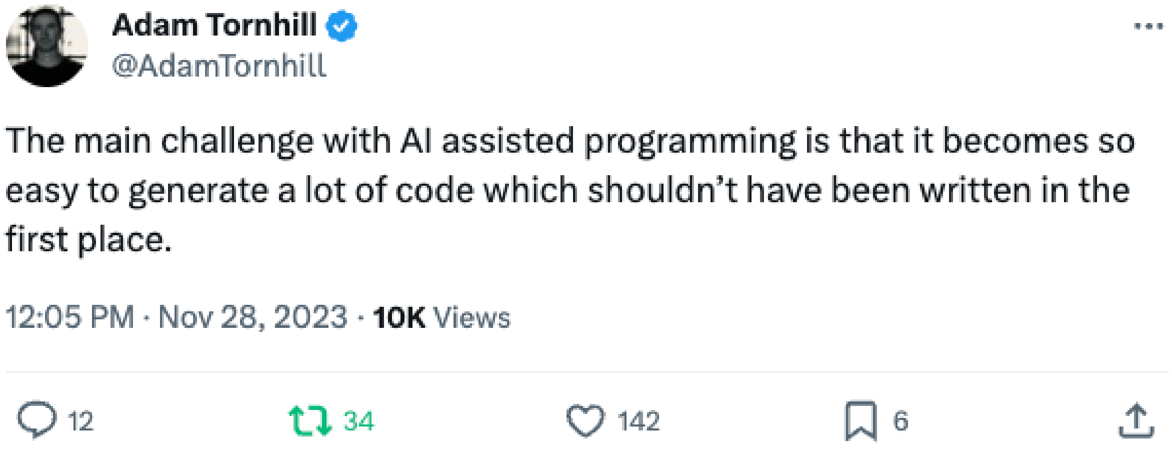
From Dohmke's 2023 blog post, "The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot"

In the same post, Dohmke asserts that more than 20,000 organizations are already using GitHub Copilot for Business. This follows GitHub's announcement from February 2023 that "more than one million people" were already using Copilot on a Personal license when Copilot for Business was released. GitHub has been making commendable progress on both advancing AI quality, and on being transparent about the result of their efforts.

What is the total percentage of developers using AI to author code? In a separate study that GitHub undertook with Wakefield Research in June 2023, they assert that "92% of U.S.-based developers working in large companies report using an AI coding tool." They go on to claim that 70% of developers say they see significant benefits to using AI. Still, an August 2023 survey by O'Reilly Publishing found that 67% of surveyed developers claimed they weren’t yet using ChatGPT *or* Copilot. This suggests that GitHub still has potential for significant market capture.

The Problem with AI-Generated Code

Developers wouldn't be adopting Copilot if they didn't believe that it accelerated their ability to produce code. GitHub's research finding on this point says "developers are 75% more fulfilled when using Copilot." To a first approximation, developers embrace the product. This doesn’t reveal whether their near-term satisfaction will be shared by those who go on to maintain the code. Initial impressions from longtime code researcher Adam Tornhill (author, Your Code as a Crime Scene) are skeptical:



Adam Tornhill
@AdamTornhill

The main challenge with AI assisted programming is that it becomes so easy to generate a lot of code which shouldn’t have been written in the first place.

12:05 PM · Nov 28, 2023 · **10K** Views

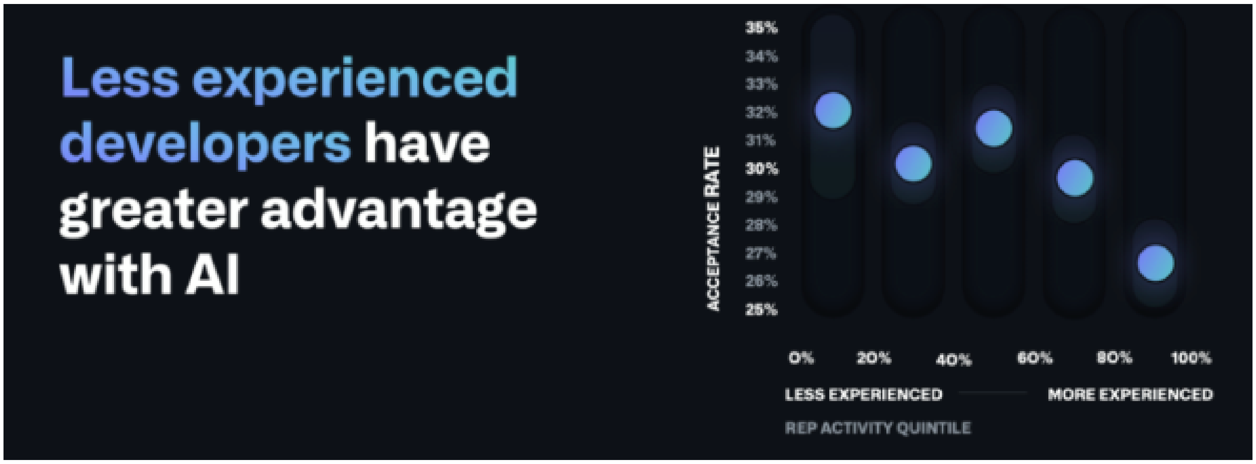
Developer researchers are concerned by the impact of AI assisted programming

GitHub claims that code is written "55% faster" with Copilot. But what about code *that shouldn't be written in the first place*? The problem here is that code spends 10x more time being **read** than being **written**, according to Robert Martin, author of Clean Code: A Handbook of Agile Software Craftsmanship. Writing bad code faster implies considerable pains for the subsequent code readers.

That is the first of many challenges facing developers who use an AI assistant. Others include:

1. **Being inundated with suggestions for added code, but never suggestions for updating, moving, or deleting code**. This is a user interface limitation of the text-based environments where code authoring occurs.
2. **Time required to evaluate code suggestions can become costly**. Especially when the developer works in an environment with multiple, competing auto-suggest mechanisms (this includes the popular JetBrains IDEs [11])
3. **Code suggestion is not optimized by the same incentives as code maintainers**. Code suggestion algorithms are incentivized to propose suggestions most likely to be accepted. Code maintainers are incentivized to minimize the amount of code that needs to be read (I.e., to understand how to adapt an existing system).

These drawbacks may explain the difference between the greater tendency of Junior Developers to accept code suggestions compared to their more experienced counterparts. According to GitHub’s research:



GitHub's own data suggests that Junior Developers use Copilot around 20% more than experienced developers

Experienced developers have the most informed understanding of how costly code will be to maintain over time. If they are more averse to using AI suggestions, it raises questions about the extra code that Junior developers are now contributing, faster than ever?

Code Change Definitions

To analyze how code quality is changing, we will review the differences in types of code changes observed in 2023 vs. the years prior, when AI was much less prevalent. GitClear classifies code changes (operations) into seven categories. The first six operations are analyzed in this research:

1. **Added code**. Newly committed lines of code that are distinct, excluding lines that incrementally change an existing line (labeled "Updates"). "Added code" also does not include lines that are added, removed, and then re-added (these lines are labeled as "Updated" and "Churned")
2. **Deleted code**. Lines of code that are removed, committed, and not subsequently re-added for at least the next two weeks.
3. **Moved code**. A line of code that is cut and pasted to a new file, or a new function within the same file. By definition, the content of a "Moved" operation doesn't change within a commit, except for (potentially) the white space that precedes the content.
4. **Updated code**. A committed line of code based off an existing line of code, that modifies the existing line of code by approximately three words or less.
5. **Find/Replaced code**. A pattern of code change where the same string is removed from 3+ locations and substituted with consistent replacement content.
6. **Copy/Pasted code**. Identical line contents, excluding programming language keywords (e.g., `end`, `})`, `[`), that are committed to multiple files or functions within a commit.
7. **No-op code**. Trivial code changes, such as changes to white space, or changes in line number within the same code block. No-op code is excluded from this research.

Specific examples of GitClear's code operations can be found in the Diff Delta documentation. GitClear has been classifying git repos by these operations since 2020. As of January 2024, GitClear has analyzed and classified around a billion lines of code over four years, from a mixture of commercial customers (e.g., NextGen Health, Verizon) and popular open source repos (e.g., Facebook React, Google Chrome). 153 million lines of code were meaningful (not No-op) line changes, used for this research.

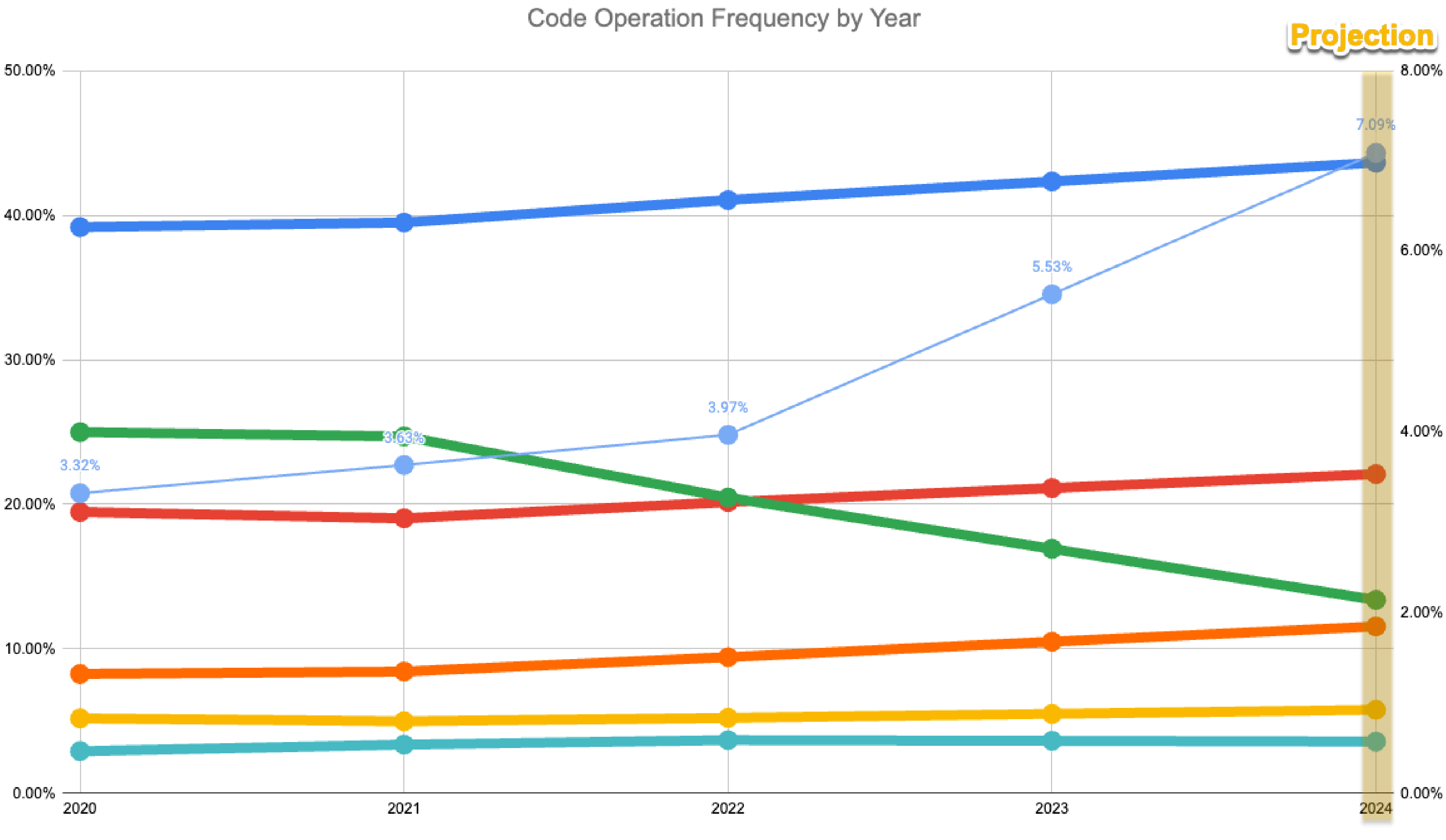
Along with the evolution of code change operations, we are also exploring the change in "**Churned code**." This is not treated as a code operation, because a churned line can be paired with many operations, including "Added," "Deleted," or "Updated" code. For a line to qualify as "churned," it must have been authored, pushed to the git repo, and then reverted or substantially revised within the subsequent two weeks. Churn is best understood as "changes that were either incomplete or erroneous when the author initially wrote, committed, and pushed them to the company's git repo."

Trends in Commit Line Operations

To understand how Copilot has changed code quality, we analyzed the number of different line operations that GitClear has observed, segmented by the year in which the code was authored (using the `authored_at` date within the git commit header [12]). The raw numbers for this analysis are included in the Appendix. Here are the percentages by year:

	Added	Deleted	Updated	Moved	Copy/pasted	Find/replaced	Churn
2020	39.2%	19.5%	5.2%	25.0%	8.3%	2.9%	3.3%
2021	39.5%	19.0%	5.0%	24.8%	8.4%	3.4%	3.6%
2022	41.0%	20.2%	5.2%	20.5%	9.4%	3.7%	4.0%
2023	42.3%	21.1%	5.5%	16.9%	10.5%	3.6%	5.5%
2024	43.6%	22.1%	5.8%	13.4%	11.6%	3.6%	7.1%

Here are how these look in graph form, where the left axis illustrates the prevalence of code change operations (which, as percentages, sum to 1). The right axis and light blue line track the corresponding change in "Churn" code:



GitClear.com Original Research

The projections for 2024 utilize OpenAI's `gpt-4-1106-preview` Assistant to run a quadratic regression on existing data. The full method used to interrogate the OpenAI Assistant is provided in the Appendix. Given the exponential growth of Copilot reported by GitHub, and AI Assistants in general, it seems likely that 2024's numbers will continue the trends that began to take form in 2022 and accelerated in 2023. Looking only at the differences in operation frequency between 2022 and 2023, we find **three red flags for code quality**:

Operation	YoY change
Added	+3.1%
Deleted	+4.8%
Updated	+5.2%
Moved	-17.3%
Copy/Pasted	+11.3%
Find/Replaced	-1.3%
Churn	+39.2%

Interpreting Code Operation Changes

The most significant changes observed in 2023 are to "Churn," "Moved," and "Copy/Pasted" code. The implications for each change are reviewed in this section.

Burgeoning Churn

Recall that "Churn" is the percentage of code that was pushed to the repo, then subsequently reverted, removed or updated within 2 weeks. This was a relatively infrequent outcome when developers authored all their own code -- only 3-4% of code was churned prior to 2023, although there is also a hint of the coming uptick in 2022, when Churn jumped 9%. 2022 was the first year Copilot was available in beta, and the year that ChatGPT became available.

In 2022-2023, the rise of AI Assistants are strongly correlated with "mistake code" being pushed to the repo. If we assume that Copilot prevalence was 0% in 2021, 5-10% in 2022 and 30% in 2023 (corresponding to citations [1] and [8]), the Pearson correlation coefficient between these variables is 0.98 (see “Correlation between Churn & Copilot” in Appendix for more details on calculation). Which is to say, that they have grown in tandem.

The more Churn becomes commonplace, the greater the risk of mistakes being deployed to production. If the current pattern continues into 2024, more than 7% of all code changes will be reverted within two weeks, double the rate of 2021. Based on this data, we expect to see an increase in Google DORA's "Change Failure Rate" when the “2024 State of Devops” report is released later in the year, contingent on that research using data from AI-assisted developers in 2023.

Less Moved Code Implies Less Refactoring, Less Reuse

Moved code is typically observed when refactoring an existing code system. Refactored systems in general, and moved code in particular, underpin code reuse. As a product grows in scope, developers traditionally rearrange existing code into new modules and files that can be reused by new features. The benefits of code reuse are familiar to experienced developers – compared with newly added code, reused code has already been tested & proven stable in production. Often, reused code has been touched by multiple developers, so is more likely to include documentation. This accelerates the interpretation of the module by developers who are new to it.

Combined with the growth in code labeled "Copy/Pasted," there is little room to doubt that the current implementation of AI Assistants discourages code reuse. Instead of refactoring and working to DRY ("Don't Repeat Yourself") code, these Assistants offer a one-keystroke temptation to repeat existing code.

More Copy/Pasted Code Implies Future Headaches

There is perhaps no greater scourge to long-term code maintainability than copy/pasted code. To an extent, when a non-keyword line of code is repeated, the code author is admitting "I didn't have the time to evaluate the previous implementation." By re-adding code (vs. reusing it), the chore is left to future maintainers to figure out how to consolidate parallel code paths that implement repeatedly-needed functionality.

Since most developers derive greater satisfaction from "implementing new features" than they do "interpreting potentially reusable code," copy/pasted code often persists long past its expiration date. Especially on less experienced teams, there may be no code maintainer with the “moral authority” to remove the duplicative code. Even when there are Senior Developers possessing such authority, the willpower cost of understanding code well enough to delete it is hard to overstate.

If there isn't a CTO or VP of Engineering who actively schedules time to reduce tech debt, you can add "executive-driven time pressures" to the list of reasons that newly added copy/paste code will never be consolidated into the component libraries that underpin long-term development velocity.

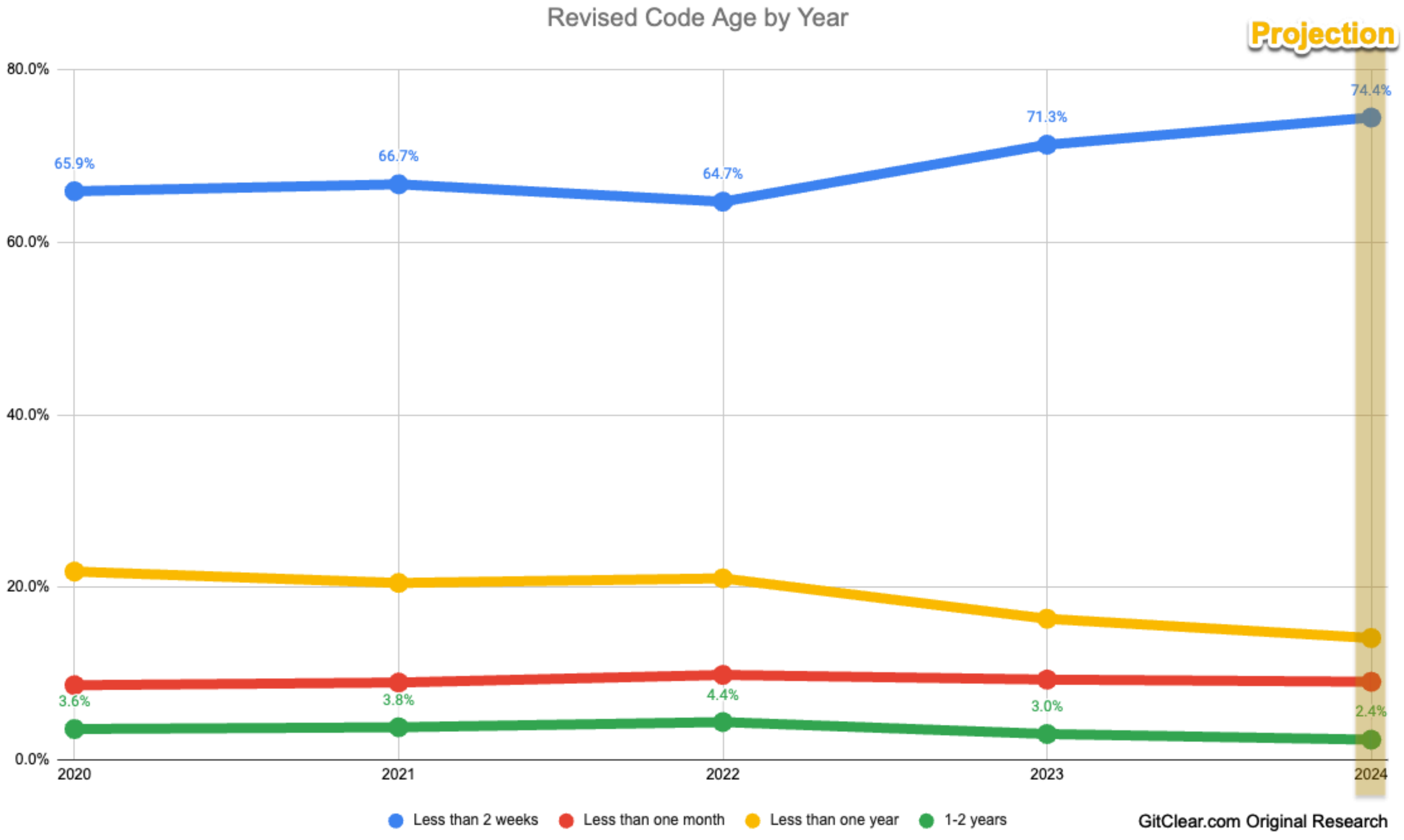
Since GitClear operations only include code that is duplicated within a single commit, it is likely that the 11% copy/paste measured in 2023 is only a fraction of the total pasting being quietly seeded into repos during 2024.

Trends in Revised Code Age

A second, independent means to assess how code quality has changed in 2023 vs. before is to analyze the data from GitClear's Code Provenance derivation. A “Code Provenance” assessment evaluates the length of time that passes between when code is authored, and when it is subsequently updated or deleted.

	Less than 2 weeks	Less than one month	Less than one year	1-2 years
2020	65.9%	8.7%	21.8%	3.6%
2021	66.7%	9.0%	20.5%	3.8%
2022	64.7%	9.9%	21.1%	4.4%
2023	71.3%	9.3%	16.4%	3.0%
2024	74.4%	9.1%	14.1%	2.4%

The corresponding raw numbers are in the Appendix. Graphing the data, we find:



Interpreting Code Age Trends

Code Provenance data corroborates the patterns observed in the Code Operation analysis. The age of the code when it is replaced has shifted much younger from 2022 to 2023. Specifically, code replaced in less than two weeks has jumped by 10%. Meanwhile, code older than one month was changed 24% less frequently in 2023 vs 2022 (19.4% of all changes vs. 25.5% previously).

The trend implies that, prior to AI Assistants, developers may have been more likely to find recently authored code in their repo to target for refinement and reuse. Around 70% of products built in the early 2020s use the Agile Methodology, per a Techreport survey [5]. In Agile, features are typically planned and executed per-Sprint. A typical Sprint lasts 2-3 weeks. It aligns with the data to surmise that teams circa 2020 were more likely to convene post-Sprint, to discuss what was recently implemented and how to reuse it in a proximal Sprint.

Questions for Follow Up Research

Can incentives be created to counteract the "add it and forget it" tab-based invocation that pervades code suggestion engines of 2024?

While AI could be trained to identify code consolidation opportunities, when would it be invoked? An alternative UI would be needed to review code deletions and updates, alongside prospective additions. Furthermore, the same executive pressures that prevent teams from scheduling time to reduce tech debt today would probably prevent them from adopting a hypothetical “code cleanup” tool. Still, if the creator of a Code Assistant is interested in exploring how to consolidate code, GitClear would like to work with them. Our contact information is in the Appendix.

Another salient question in light of this data: at what rate does development progress become inhibited by additional code? Especially when it comes to copy/pasted code, there is almost certainly an inverse correlation between "the number of lines of code in a repo" and "the velocity at which developers can modify those lines." The current uncertainty is "when is the accumulated copy/paste tech debt too great to ignore?" Knowing the rate at which slowdown takes hold would allow future tools to highlight when a manager should consider cutting back time on new features.

A final question worthy of exploration: what is the total percentage of copy/pasted code that is now occurring, compared to 2020-2022? Since GitClear currently measures only copy/paste code within the context of an individual commit, it seems likely that the total copy/paste volume (all non-keyword, non-comment lines of code repeated within a file) might be double what GitClear currently measures. Could copy/paste really represent 20-25% of all code operations in 2024?

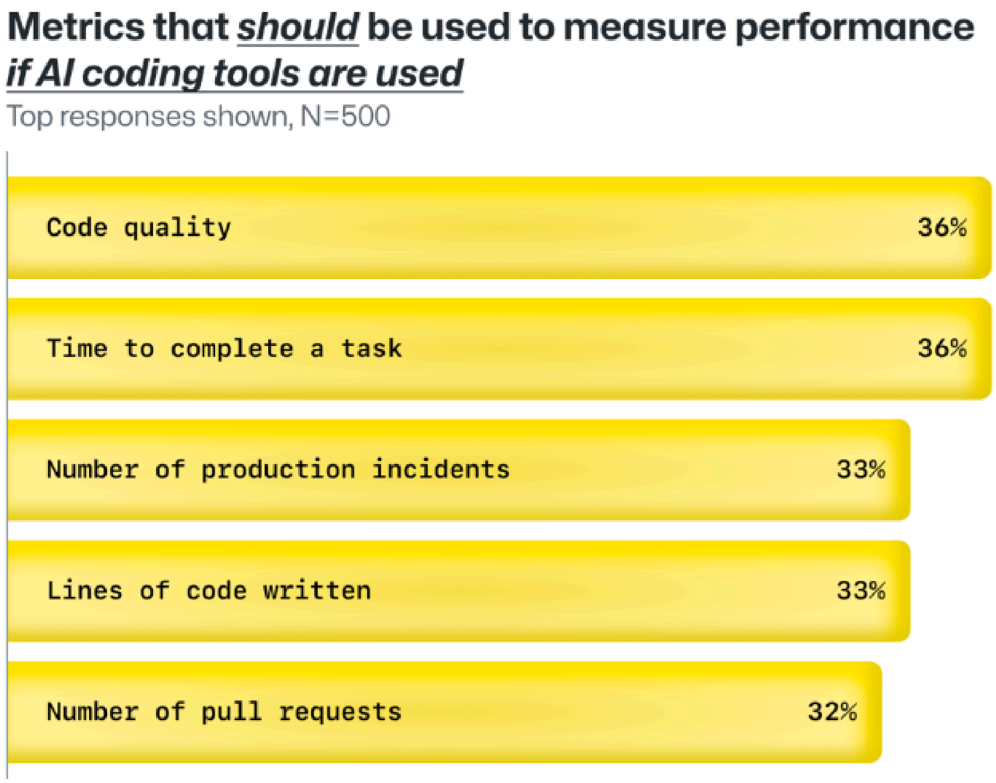
GitClear will look to address these questions in future research, and we encourage other researchers in the field to contribute their data. If you would like to partner with GitClear to undertake further research, our contact information is in the Appendix.

Conclusion: Devs Wary for a Reason?

By both data points we evaluated, negative pressures on code quality were present in 2023. This correlates with the proliferation of LLMs in general, and AI Code Assistants in particular.

Developer assessments, like GitHub's 2023 survey with Wakefield Research, hint that developers already perceive the decrease in code quality. When asked "What metrics *should* you be evaluated on, **absent AI**," their top response was "Collaboration and Communication," followed by "Code Quality" in second place.

When the question switched to "What metrics should you be evaluated on, **when actively using AI?**" their responses shifted, with "Code Quality" now the top concern, and "Number of Production Incidents" rising to the #3 concern:



From GitHub's Survey on AI Impact

While individual developers lack the data to substantiate *why* "Code Quality" and "Production Incidents" become more pressing concerns with AI, our data suggests a possible backstory: When developers are inundated with quick and easy suggestions that will work in the short term, it becomes a constant temptation to add more lines of code without really checking whether an existing system could be refined for reuse.

To the extent that inexperienced developers continue to be offered implicit copy/paste suggestions via the tab key, the fix for this situation won't be easy. It is beholden on engineering leaders to monitor incoming data and consider its implications for future product maintenance. Developer Analytics tools, including GitClear, can help detect the rate at which problematic code is being seeded. Specific questions to evaluate:

1. Is the percentage of code reuse falling?
2. Are there changes in how much code is moved and copy/pasted?
3. Is it easy for developers to discover code reuse opportunities?

Further discussion on how GitClear can address these questions [A3].

How will AI Assistants and Copilot transform what it means to be a developer? There's no question that, as AI has proliferated, we have entered an era where code lines are added faster than ever. The better question for 2024: who's on the hook to clean up what’s left afterward?

Citations

1. The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot [GitHub]
2. GitHub Copilot for Business is now available [GitHub]
3. Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle [GitHub]
4. Diff Delta and Commit Groups [GitClear]
5. Techreport survey: 71% of teams use Agile [Techreport]
6. What is "code provenance" and why does it matter? [GitClear]
7. Survey reveals AI's impact on the developer experience [GitHub]
8. The next generation of developer productivity [O'reilly]
9. Your Code as a Crime Scene [Pragmatic Programmers]
10. Clean Code: A Handbook of Agile Software Craftsmanship and specific quote cited [Robert C. Martin, author]
11. JetBrains AI: Supercharge your tools. Embrace new freedom [JetBrains]
12. https://git-scm.com/docs/git-commit [Git docs]
13. Using the Tech Debt Browser [GitClear]
14. Don't repeat yourself [Wikipedia]
15. X: Tweet from Adam Tornhill [X/Twitter]

Appendix

Data used to build this research is included below.

A1: Raw data for changed line counts

	Added	Deleted	Updated	Moved	Copy /pasted	Find/ replaced	Lines changed	Churn
2020	9,071,731	4,508,098	1,202,480	5,786,718	1,911,855	676,000	23,156,882	769,493
2021	14,464,864	6,969,778	1,826,579	9,043,649	3,087,530	1,234,213	36,626,613	1,331,278
2022	16,868,378	8,280,031	2,146,768	8,407,677	3,873,240	1,512,708	41,088,802	1,630,703
2023	22,626,714	11,288,962	2,938,800	9,040,659	5,607,373	1,942,194	53,444,702	2,952,912
2024	28,708,803	14,535,353	3,803,275	8,804,121	7,599,970	2,355,662	65,800,602	4,665,263

Here were some secondary characteristics of the data set analyzed, to aid in evaluating its validity/applicability relative to existing data sets the reader may possess:

Year	Commit count	Committer count	Repos analyzed	Code files changed
2020	381347	12761	497	1368549
2021	623264	17577	643	2207498
2022	723823	18446	993	2616263
2023	1019680	21700	1294	3414136

In CSV pasteable form, for your reanalysis convenience (2024 omitted since it is a projection you can replace with your own):

```
Year,Added,Deleted,Updated,Moved,Copy/pasted,Find/replaced,Lines changed,Churn
2020,9071731,4508098,1202480,5786718,1911855,676000,23156882,769493
2021,14464864,6969778,1826579,9043649,3087530,1234213,36626613,1331278
2022,16868378,8280031,2146768,8407677,3873240,1512708,41088802,1630703
2023,22626714,11288962,2938800,9040659,5607373,1942194,53444702,2952912
```

Queries used to produce data

The data was stored in a Postgres database and was queried via Ruby on Rails' ActiveRecord.

```
# Operation by year
2020.upto(2023).map { |year| CodeLine.where(authored_at: Time.new(year, 1,
1)..Time.new(year + 1, 1, 1), commit_impacting:
true).group(:operation_em).count }
# Commits by year
Commit.impacting.where(authored_at:
Time.local(2020,1,1)..Time.local(2024,1,1)).group("EXTRACT (year from
authored_at)").count
# Committers committing by year
2020.upto(2023).map { |year|
Committer.joins(:commits).merge(Commit.impacting).where(commits: {
authored_at: Time.local(year,1,1)..Time.local(year+1,1,1)
}).group(:id).count.size }
# Repos changed by year
2020.upto(2023).map { |year|
Repo.joins(:commits).merge(Commit.impacting).where(commits: { authored_at:
Time.local(year,1,1)..Time.local(year+1,1,1) }).group(:id).count.size }
# Files by year
2020.upto(2023).map { |year|
CommitCodeFile.impacting.joins(:commit).where(commits: { authored_at:
Time.local(year,1,1)..Time.local(year+1,1,1) }).group(:id).count.size }
```

A2: Largest known database of structured code data

As of January 2024, there is no publicly available dataset that indexes code changes. Published data on code stats originates only from the handful of companies that hold or evaluate code data. Among these companies, including GitHub, GitLab, Bitbucket, Azure Devops, GitKraken, and SonarQube, all classify code changes as a binary “add” or “delete” [example]. GitClear is the only company thus far that recognizes a broader set of operations:

1. Added code
2. Updated code
3. Deleted code
4. Copy/pasted code
5. Find/replaced code
6. Moved code
7. No-op code

GitClear’s data is split about two-thirds private corporations that have opted in to anonymized data sharing, and one-third open source projects (mostly those run by Google, Facebook, and Microsoft).

In addition to the code operation data, GitClear's data set also segments and excludes lines if they exist within auto-generated files, subrepo commits, and other exclusionary criteria enumerated in this documentation. As of January 2024, that documentation suggests that a little less than half of the "lines changed" by a conventional git stats aggregator (e.g., GitHub) would qualify for analysis among the 150m lines in this study. The study does include commented lines – future research could compare comment vs. non-comment lines. It could also compare "test code" vs "other types of code," which probably influences the levels of copy/paste.

If you know of other companies that report code operations of comparable granularity, please contact hello@gitclear.com and this section will be updated, and a new PDF document will be uploaded with credit given to the contributor (if desired).

A3: GitClear solutions

GitClear offers reports that answer all three questions (operations recognized, operation report, provenance report). It also offers a Tech Debt browser, and the ability to enable email notifications when quality is on the downward slope.

Raw data for revised line counts

Year	Less than 2 weeks	Less than one month	Less than one year	1-2 years
2020	550362	72471	182420	30074
2021	891008	120029	274125	50825
2022	1136604	173370	369925	77463
2023	1941351	254082	445869	82405

In CSV pasteable form, for your own reanalysis convenience:

```
Year,Less than 2 weeks,Less than one month,Less than one year,1-2 years,Sum
2020,550362,72471,182420,30074,835327
2021,891008,120029,274125,50825,1335987
2022,1136604,173370,369925,77463,1757362
2023,1941351,254082,445869,82405,2723707
```

Queries used to produce data

The data was stored in a Postgres database and was queried via Ruby on Rails' ActiveRecord.

```
# ruby
```

```
# Revision (provenance) query
2020.upto(2023).map { |year| CodeLine.where(authored_at: Time.new(year, 1,
1)..Time.new(year + 1, 1, 1), commit_impacting:
true).group(:provenance_em).count }
```

OpenAI Assistant Data Projection

User

Looking only at the data for 2022 and 2023, what would a quadratic regression predict as the values for each column in 2024?

Run instructions

You are a statistical interpreting assistant that can help interpret and project the provided CSV file

```
code_interpreter(import pandas as pd # Load the CSV file to check its structure and contents such as colu… )
↳ Year Added Deleted Updated Moved Copy/pasted Find/replaced Churn 0 2020 39.18% 19.47% 5.19% 24.99…
```

Provenance interpreter

The data structure shows that the CSV file contains percentages for each column along with the corresponding year. To move forward with the analysis, the next step is to:

1. Filter the data for the years 2022 and 2023.
2. Remove the percent signs and convert the values to floats.
3. Perform a quadratic regression for each column based on the filtered data.
4. Predict the values for each column for the year 2024.

I will begin by filtering the data and converting the percentages to numeric values.

```
code_interpreter(# Filter the dataset for years 2022 and 2023 only df_filtered = df[df['Year'].isin([2022… )
↳ Year Added Deleted Updated Moved Copy/pasted Find/replaced Churn 2 2022 0.4105 0.2015 0.0522 0.20…
```

Provenance interpreter

The data for 2022 and 2023 has been filtered and the values have been converted to float type with the percent signs removed. Now, I'll proceed with performing a quadratic regression for each column to predict the values for 2024.

```
code_interpreter(from numpy import polyfit, poly1d # Prepare the years and the prediction year years = df… )
↳ /home/sandbox/.local/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3508: RankWarni…
```

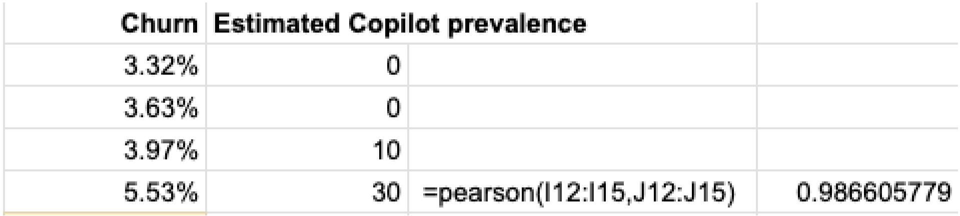
Provenance interpreter

Based on a quadratic regression using only the data from 2022 and 2023, the predicted values for each column in 2024 are as follows:

- Added: 43.63%
- Deleted: 22.09%
- Updated: 5.78%
- Moved: 13.38%
- Copy/pasted: 11.55%
- Find/replaced: 3.58%
- Churn: 7.09%

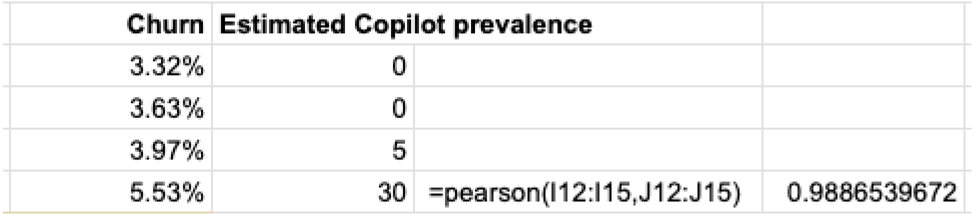
These values are expressed as percentages, rounded to two decimal places. Keep in mind that the quality of these predictions depends on the patterns present in the data for 2022 and 2023 and assumes that the same trends will continue in a quadratic manner. Additionally, a quadratic regression using just two data points can lead to large errors if the underlying process is not well-represented by this model.

Correlation between Churn and estimated Copilot prevalence



Churn	Estimated Copilot prevalence		
3.32%	0		
3.63%	0		
3.97%	10		
5.53%	30	=pearson(I12:I15,J12:J15)	0.986605779

The first year is 2020, the last year is 2023. The estimate of 30% is taken from the independent O'Reilly Publishing survey in citation [8], stating that less than a third of developers claim to currently use Copilot or ChatGPT for coding as of August 2023. The 10% estimate is based on GitHub's blog post in [1] stating that Copilot launched a year earlier, and had grown to include "millions of developers." If we guess that the prevalence of Copilot was only 5% in 2022, there is no significant change to the Pearson coefficient.



Churn	Estimated Copilot prevalence		
3.32%	0		
3.63%	0		
3.97%	5		
5.53%	30	=pearson(I12:I15,J12:J15)	0.9886539672

Updates

- Jan 26, 2024 Improved clarity and consonance of data with language. Add Contact Information.

Contact information

If you would like to discuss this research, or have ideas on how to improve it, please contact hello@gitclear.com or bill@gitclear.com directly. We are happy to consider improvements to the clarity of this writing, or to explain how GitClear can help teams measure the metrics explored by this research.