Case study: How much does it cost to maintain a copy of an open source software package (at Google)?

Author: Sophia Vargas, Google
Date published: September, 2026

Overview:
Open source software is used pervasively, with some estimates that over 90% of codebases contain open source software.[1] While many adopt open source software because it's free and thus cheaper than writing a proprietary alternative, the actual costs to use open source software are non-zero. While understanding the full cost to maintain open source software is a much larger question and area of research, for this study we focused on the cost to maintain *internal vendored copies that may include local patches*.[2] We hope that our findings will help bolster research on the valuation of open source software.

As this study was completed at Google, we acknowledge the inherent bias and limitations of our findings. The majority of code at Google is organized as a monorepo and open source software packages are imported (copied) into our //third_party library.[3][4] Barring exceptions, there is only one version of any open source package available in //third_party. Note that Google uses a variety of tooling (such as copybara) to manage packages in //third_party.[5][6] Within this study, we did not dictate what tools (including GenAI) could or could not be used in the update process, and as such did not investigate the impact of any specific tool.

Methodology:
To understand the costs associated with maintaining open source software packages inside of Google, we focused on the following questions:

> *RQ1: How long does it take for an engineer (not a maintainer of the upstream project) to update an open source software package at Google?*
>
> *RQ2: Can complexity metrics predict the relative effort/amount of time to update a package at Google?*

To answer these questions, we designed a multi-modal research study:

---

[1] https://static.carahsoft.com/concrete/files/1617/1597/8665/2024_Open_Source_Security_and_Risk_Analysis_Report_WRAPPED.pdf
[2] https://ieeexplore.ieee.org/abstract/document/7816452
[3] https://research.google/pubs/why-google-stores-billions-of-lines-of-code-in-a-single-repository/
[4] https://opensource.google/documentation/reference/thirdparty
[5] https://opensource.google/documentation/reference/thirdparty/oneversion
[6] https://github.com/google/copybara

1) Survey: We ran an internal survey of open source users and contributors at Google in October-November 2024 and included the following questions:

Survey:

- *On average, how long does it take to update a package in //third_party? (multiple choice)*
- *In your experience, what characteristics of a //third_party package or the update process make it take longer than average? (select all that apply)*

2) Direct data collection (DDC): For ease in reference, we will refer to this sample as DDC. We collected data directly from software engineers as they performed 54 package updates between April-December 2024, which represented less than 1% of package updates completed during that time period. Packages were selected based on internal priorities and the majority (52/54) were written in Go. For each update, each engineer completed the following form to capture their experience:

DDC:

- **Personal bias:**
  - **Language competency**: *What is your familiarity with the language of the package? (1 - Not familiar to 5 - Very familiar)*
  - **Project awareness**: *What is your familiarity with the project/package? (1 - Not familiar to 5 - Very familiar)*
  - **Usage awareness**: *How familiar are you with how this package is used / has been customized within the company? (1 - Not familiar to 5 - Very familiar)*
- **Effort**: (in hours)
  - **Time to onboard**: *How much time did you need to learn about the package and understand how to complete an update?*
  - **Time to update package**: *How much time did it take you to update the package—including any required tooling/infrastructure setup?*
  - **Time to remediate:** *How long did it take to resolve any/all downstream dependencies and breakage?*
- **Post mortem**: (Open ended) *Did anything break? If so please describe*.

3) Complexity metrics assessment: Using a combination of existing literature, and a review of available data sources, we curated a list of metrics to test metadata about each package against DDC (see Table 1). Note that this dataset was not complete—for some packages we were only able to collect a subset of these metrics (see case counts (N) within results below).

Table 1: Complete Metrics Assessed

Feature	Metrics	Data source(s)	Notes
Age: how stale is our version?	# of days/time since last update # of days/time since last installed version	Internal source	
Upstream velocity: what is the rate of change?	# of GitHub issues in 2024 # of GitHub PushEvents (PE) and PullRequestEvents (PR) in 2024	githubarchive.month.2024*	Other metrics considered but excluded due to lack of readily available data at scale: # of releases per year # commits per year # Lines-of-Code (LOC) added/removed per year
Size of package	# of files in //third_party # of upstream contributors	Internal source; githubarchive.month.2024*	Other metrics considered but excluded due to lack of readily available data at scale: # LOC # of maintainers
Internal usage	# of internal dependencies ON package	Internal sources	
External dependencies	# of direct and indirect dependencies for version	bigquery-public-data.deps_dev_v1.DependenciesLatest	
Type of project	What kind of package/project is it?	Manual effort[7]	This metric was abandoned midproject due to lack of scalability
Fork debt	Patch detected (binary)	Custom script; Survey	There are more sophisticated methods to assess the complexity of custom patches but the resources required for that analysis were not in scope for this project.

[7] https://archive.fosdem.org/2025/schedule/event/fosdem-2025-5264-do-we-need-another-open-source-software-taxonomy-/

Results:

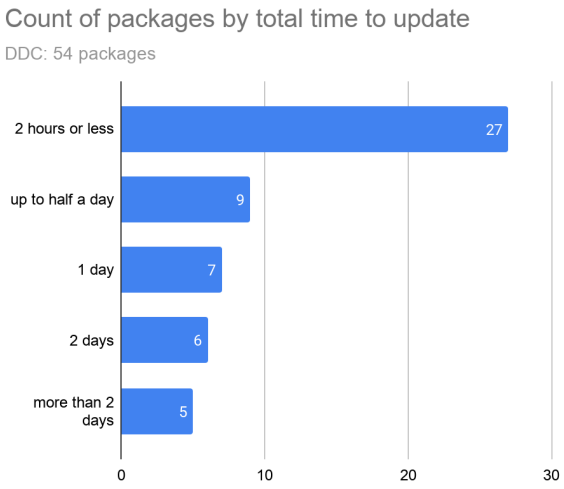
> *RQ1 (effort):* **Most engineers required 4 hours or less to update an open source software package at Google**

RQ1: For the 54 packages in DDC, **half of these updates took 2 hours or less.** For updates that took more than 2 days, the majority of time was spent on remediation (discussed further in Limitations below).

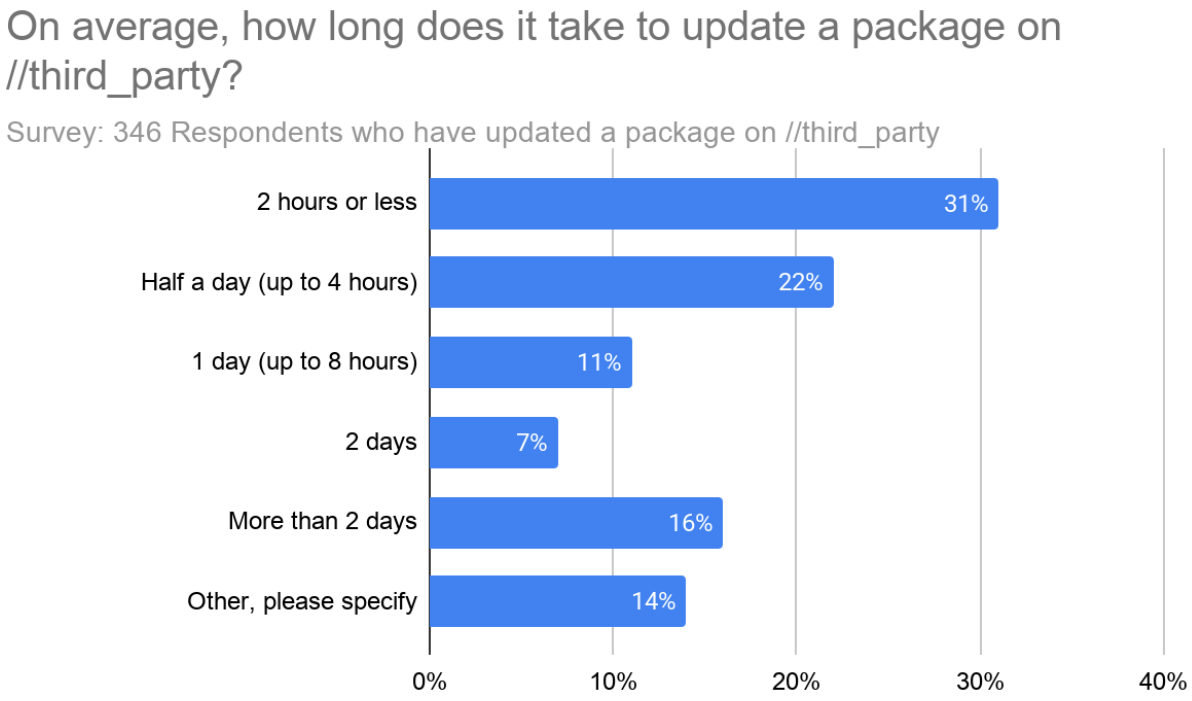
For 96% of packages in scope (52/54), engineers were very familiar with the language but completely unfamiliar with the package or how it was used inside of Google. Engineers spent minimal time onboarding (less than 1 hr on average, and a median of 0 hours) (see table 2).

Table 2: Time to update: split by phase (hours)

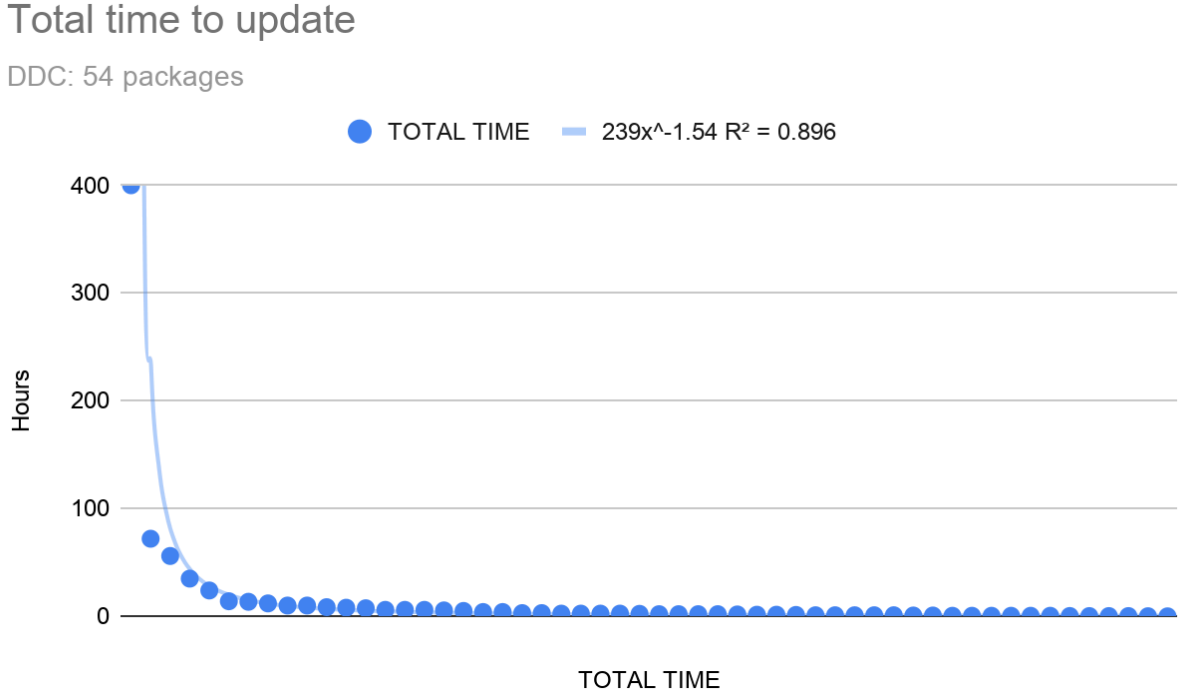
Phase	Average	Median	Min	Max
Onboard	0.4	0	0	10
Update	5.4	1.5	0.1	120
Remediate	9.4	0	0	280
Total	**13.9**	**2.2**	**0.1**	**400**



Comparing DDC results with our survey, 31% of our respondents said it took ‘2 hours or less’ to update a package in //third_party.



Our combined data indicate that the majority of updates should take 4 hours or less, however the long tail could be days to months, with one update estimated at 400 hours. As anticipated, the raw data from DDC followed a power series:



> *RQ2 (metrics):* **Our results suggest that level of effort may be indicated by custom patches, external dependency volume, and project size, while package age and internal usage levels did not appear to have a consistent impact on update complexity.**

RQ2: For each metric in scope, we reviewed Pearson correlations for every available pair in DDC. We also binned raw data from DDC and associated it with the time buckets tested in our survey, reviewing the mean, median and standard deviation to determine if we could infer baseline metrics that would indicate if packages may take 2 or more days to update.

Our results indicated that some of these metrics could be better indicators of update complexity. Based on our analysis below, we applied the following ranks (see Table 3):

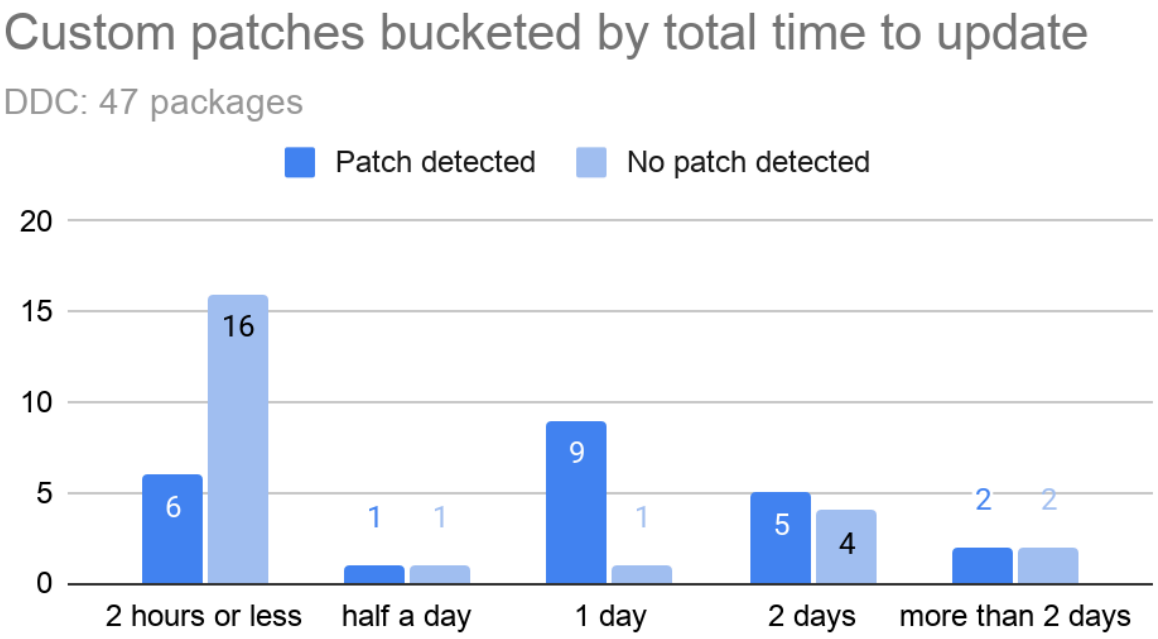
Table 3: Complete Metrics Assessment Results

Rank	Category	Metric reviewed	DDC Pearson Correlations	DDC Binned: Baseline inferred	Notes
1	**Fork debt**	Patch Detected	n/a	n/a	The median time to update was 3x for packages with custom patches compared to those without. This feature was ranked 1st in the survey
2	**Size of package**	# of files in //third_party	R(49) = 0.27	Yes	
		# of upstream contributors	R(40) = 0.89	Yes	
3	**External dependencies**	# of direct and indirect dependencies for version	R(47) = 0.91	Yes	
4	**Upstream velocity**	# of PushEvents and PullRequestEvents in 2024	R(40) = 0.49	Yes	
		# of issues in 2024	R(40) = 0.67	Yes	

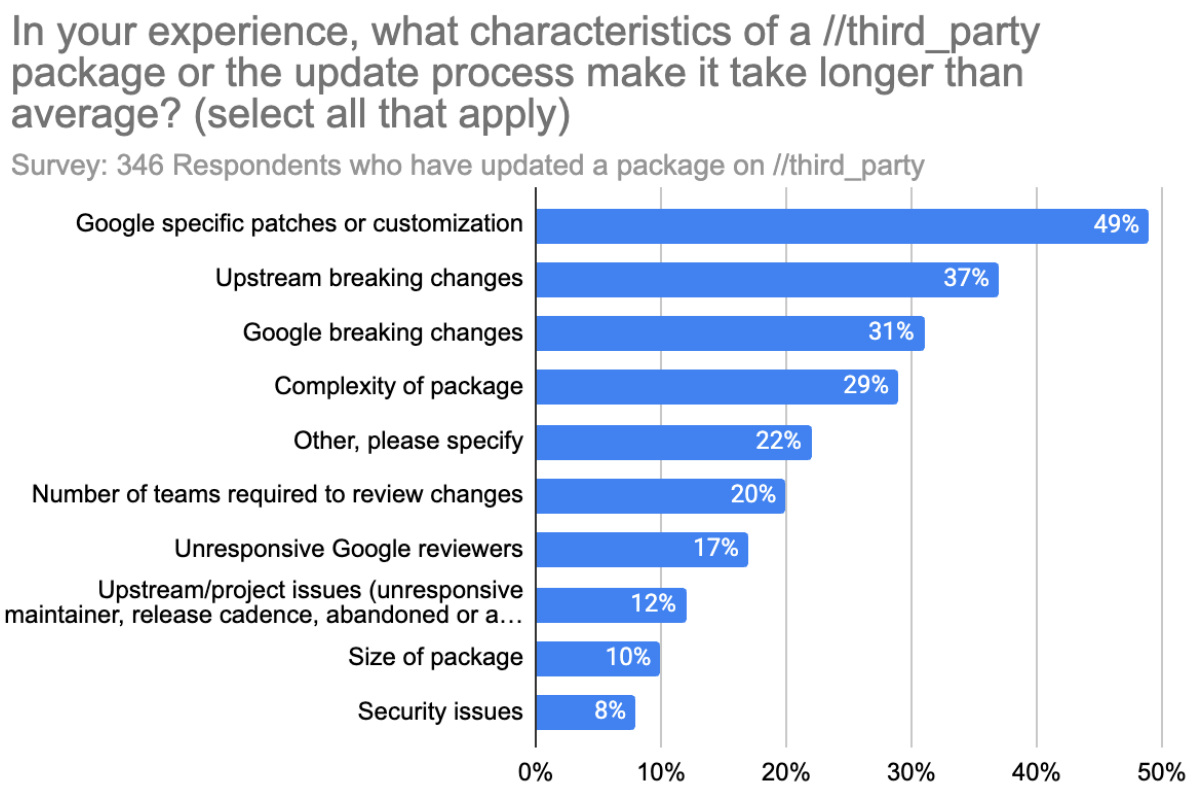
Rank	Category	Metric reviewed	DDC Pearson Correlations	DDC Binned: Baseline inferred	Notes
5	**Internal usage**	# of internal dependencies ON package	R(52) = 0.41	No	
6	**Package age**	# of days/time since last update	R(52) = -0.22	No	Survey indicated ‘breaking changes’ as the second ranked selection
		# of days/time since last installed version	R(52) = -0.18	No	
n/a	**Type of project**	n/a	n/a	n/a	

1. Fork debt: patch detected (N=47)

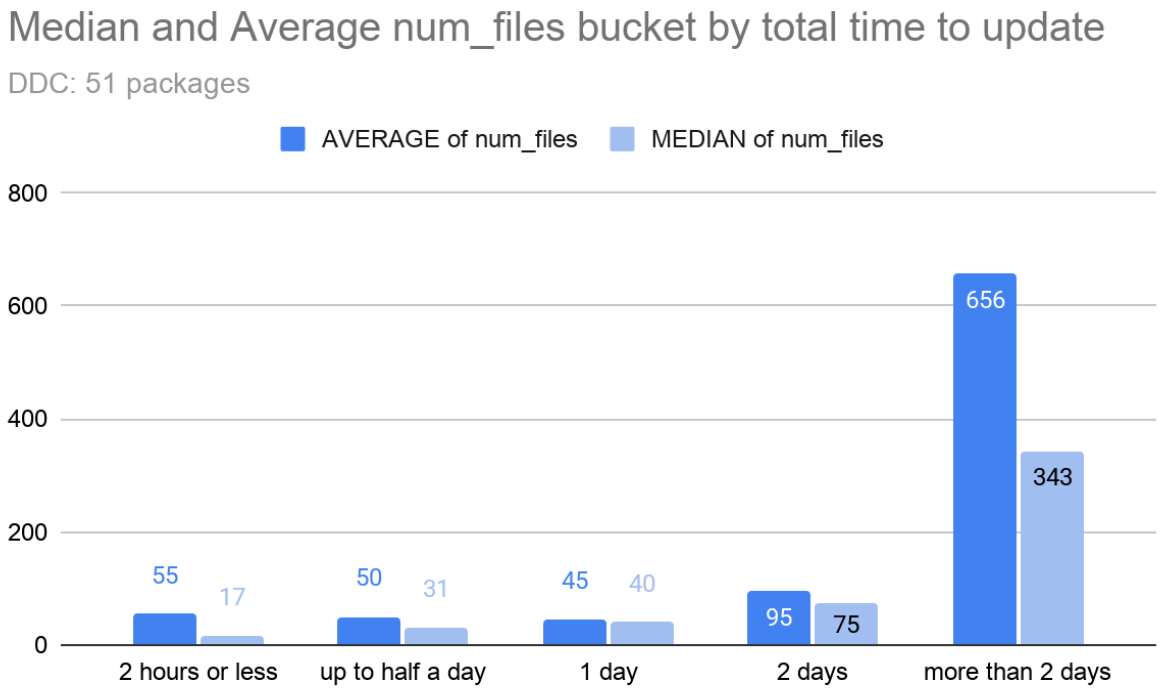
Using a custom script, we were able to detect patch files in 23 packages, 24 without patch files and 7 yielding null results in DDC. Excluding null results (47 packages remaining): the average total time to update was the same for packages with and without patches, however the median time was 3x for packages with custom patches compared to those without.



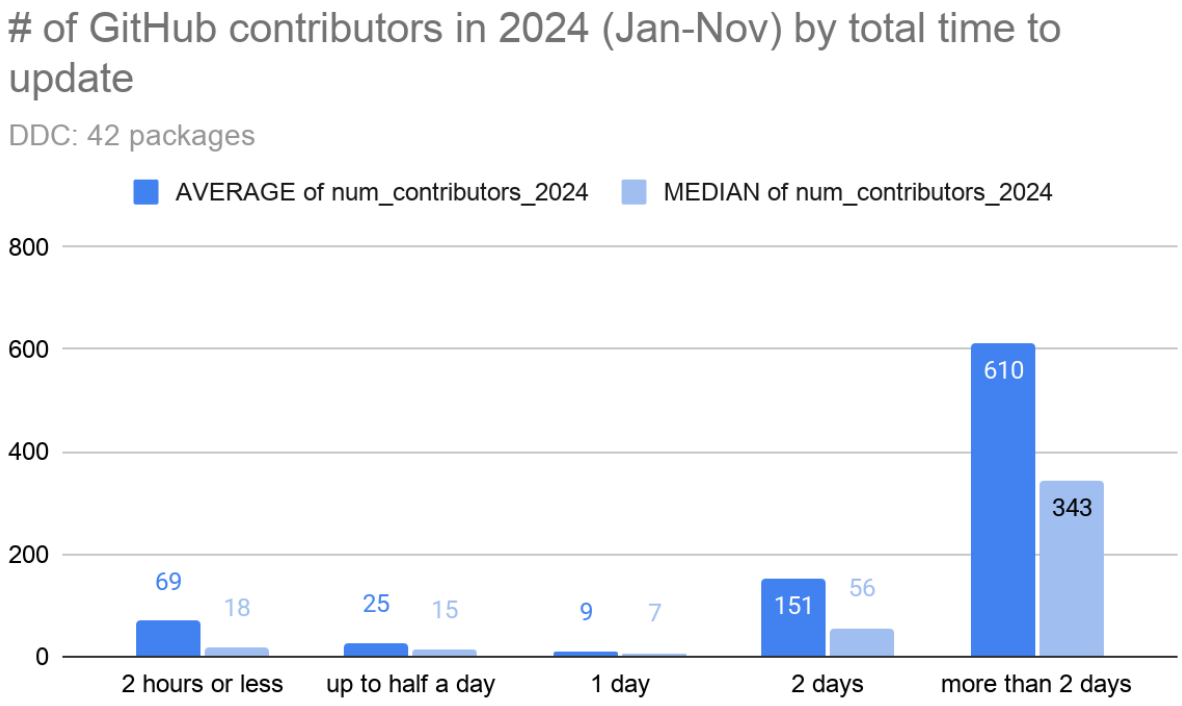
Our survey results also indicated that “Google patches and customization” were the leading characteristic of a package that makes it take longer than average to update:



2. Size of package: # of files in //third_party (N=51); # of upstream contributors (N=42)
The bucketed results from collected data show a positive correlation with the number of files in //third_party (see figure below), however a Pearson correlation test did not yield significant results.

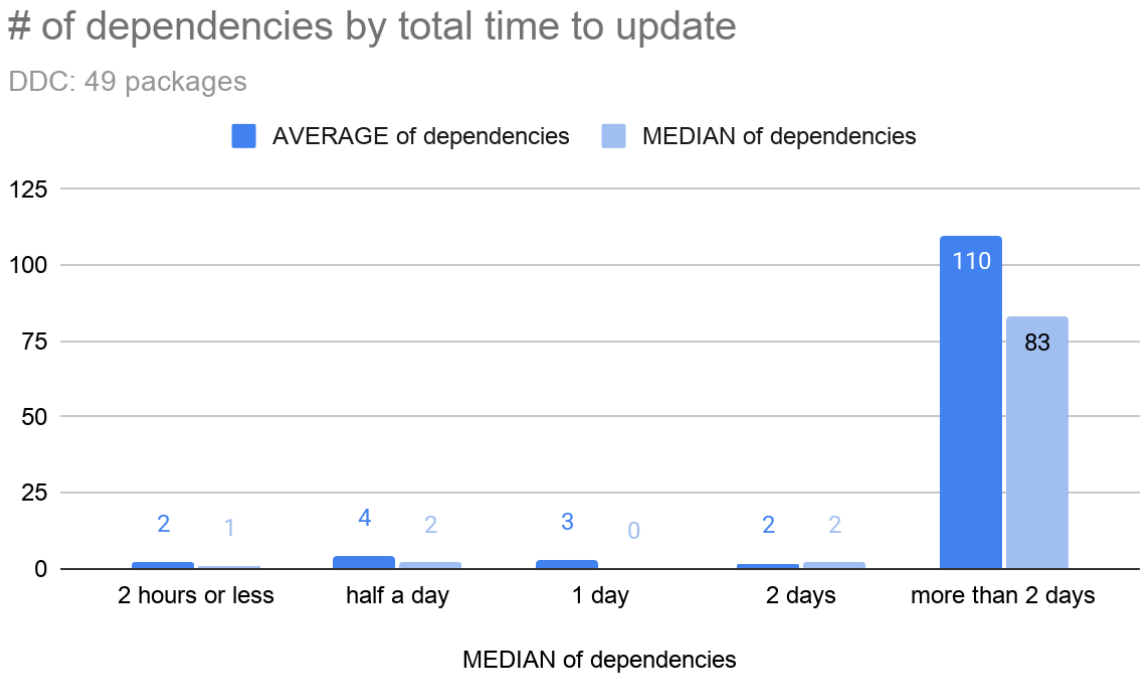


The number of contributors was more variable when reviewed in buckets, but the data collected showed a strong positive correlation with total time (r(40)=0.89).

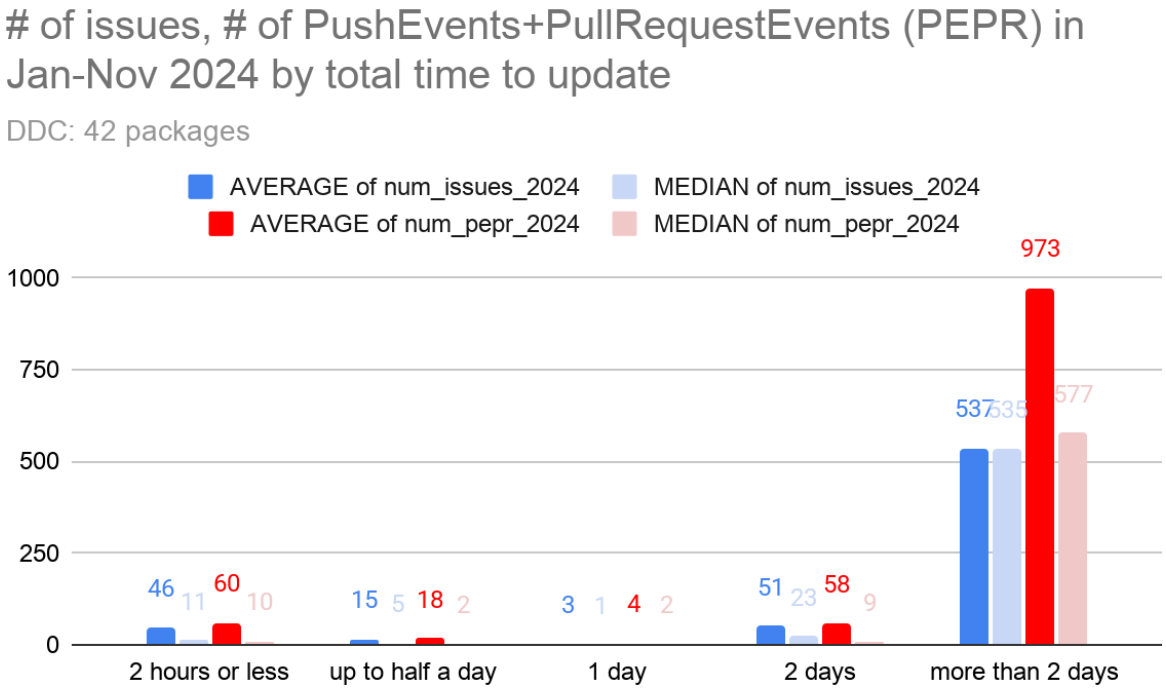


We note that this analysis differs with the ranking of ‘package size’ implied by the survey respondents. For possible explanations on this discrepancy, we note that the selected metrics to evaluate this feature are proxies for complexity that hint at size, but are not the actual size of a package (say in bytes or lines of code). As we did not define ‘package size’ in the survey, respondents could have had variable interpretations. We conclude that these metrics (number of files and number of contributors) may be better indicators of update complexity than actual package size.

3. External dependencies: # of direct and indirect dependencies for version (N=49)
While bucketing results showed that packages that took more than 2 days to update were more likely to have larger dependency counts, a Pearson correlation yielded a strongly positive correlation, suggesting this may be a useful indicator in update complexity (r(47)=0.91).



4. Upstream velocity: # of issues (N=42); # of PushEvents and PullRequestEvents (N=42)
In the small sample of 5 packages that took more than 2 days to update, all upstream repositories had larger than average number of issues and code change related activity, suggesting this may be an indicator of more complex updates. Number of issues was also moderately positively correlated with total update time (r(40)=0.67).



5. Internal usage: # of internal dependencies ON package (N=54)
The metrics used to evaluate this feature showed mixed results with none strong enough to report as a trend.

6. Age: time since last update (N=54)
In our samples, we did not observe any significant relationship between time to update and age of the current version. While upstream breaking changes can significantly impact update complexity, this was not detectable by package age or version alone.

Not evaluated: Type of project
At the time of analysis, there was no existing data source or trusted scalable way to categorize packages by project category.

Discussion

For the data collection effort described above, in most cases the engineer performing the update did not have any familiarity with the package or internal usage context outside of the underlying language. To supplement our findings, we conducted a series of interviews with other engineers who had also updated packages in //third_party. For individuals that had extensive experience with and current contribution to the upstream project, update experiences were generally more efficient and streamlined. However, there was still a wide range of effort by case. Everyone involved in this project acknowledged that updates to //third party “Vary significantly. Best case, under 2 hours. Worst case, several months”.

Limitations in this research

In addition to the Google-specific contexts enumerated above, our collected data had significant bias towards familiarity with the underlying language, more specifically Go. Given the lack of variety in our sample, we could not investigate the impact of these biases. Other internal efforts indicated that language may significantly impact maintenance efforts. We encourage this line of investigation for future studies.

Additionally, we did not detect any overarching trend across cases of extended remediation (time to update = more than 2 days). Most explanations centered on Google specific tooling and processes and as such these details cannot be shared externally.

Please note that this effort did not include significant data collection to address stale and archived packages, as well as edge cases where the update patterns were non-uniform due to upstream processes and policies.