Noname manuscript No.
(will be inserted by the editor)

An empirical study of same-day releases of popular packages in the npm ecosystem

Filipe R. Cogo · Gustavo A. Oliva · Cor-Paul Bezemer · Ahmed E. Hassan

Received: date / Accepted: date

Abstract Within a software ecosystem, client packages can reuse provider packages as third-party libraries. The reuse relation between client and provider packages is called a dependency. When a client package depends on the code of a provider package, every change that is introduced in a release of the provider has the potential to impact the client package. Since a large number of dependencies exist within a software ecosystem, releases of a popular provider package can impact a large number of clients. Occasionally, multiple releases of a popular package need to be published on the same day, leading to a scenario in which the time available to revise, test, build, and document the release is restricted compared to releases published within a regular schedule. In this paper, our objective is to study the same-day releases that are published by popular packages in the npm ecosystem. We design an exploratory study to characterize the type of changes that are introduced in same-day releases, the prevalence of same-day releases in the npm ecosystem, and the adoption of same-day releases by client packages. A preliminary manual analysis of the existing release notes suggests that same-day releases introduce non-trivial changes (e.g., bug fixes). We then focus on three RQs. First, we study how often same-day releases are published. We found that the median

✉ Filipe R. Cogo, Gustavo A. Oliva, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL), School of Computing
Queen's University, Kingston, Ontario, Canada
E-mail: {cogo,gustavo,ahmed}@cs.queensu.ca

Filipe R. Cogo
Departamento de Computação
Universidade Tecnológica Federal do Paraná, Campo Mourão, Brazil

Cor-Paul Bezemer
Analytics of Software, GAmes And Repository Data (ASGAARD) Lab
Department of Electrical and Computer Engineering
University of Alberta, Canada
E-mail: bezemer@ualberta.ca

proportion of regularly scheduled releases that are interrupted by a same-day release (per popular package) is 22%, suggesting the importance of having timely and systematic procedures to cope with same-day releases. Second, we study the performed code changes in same-day releases. We observe that 32% of the same-day releases have larger changes compared with their prior release, thus showing that some same-day releases can undergo significant maintenance activity despite their time-constrained nature. In our third RQ, we study how client packages react to same-day releases of their providers. We observe the vast majority of client packages that adopt the release preceding the same-day release would also adopt the latter without having to change their versioning statement (implicit updates). We also note that explicit adoptions of same-day releases (i.e., adoptions that require a change to the versioning statement of the provider in question) is significantly faster than the explicit adoption of regular releases. Based on our findings, we argue that (i) third-party tools that support the automation of dependency management (e.g., Dependabot) should consider explicitly flagging same-day releases, (ii) popular packages should strive for optimized release pipelines that can properly handle same-day releases, and (iii) future research should design scalable, ecosystem-ready tools that support provider packages in assessing the impact of their code changes on client packages.

Keywords Software Ecosystem, Release Management, Same-day Release, Dependencies

1 Introduction

Occasionally, multiple releases of a software project need to be published on the same day. Prior studies associate those same-day releases with a timely response to an issue that was caused by a prior release (Adams et al., 2015; Kerzazi and Adams, 2016). Same-day releases are especially relevant in software ecosystem platforms, where code reuse is enabled by means of dependencies between packages. If a package c specifies a dependency $D_{c,p}$ on another package p, then c can reuse the code of p. In this study, we refer to c as a client package and p as a provider package, respectively. In modern software ecosystems, many client packages update the adopted provider's version automatically, due to the usage of version range statements (i.e., the default npm versioning mechanism, which allows a client package to set a range of provider versions that are automatically updated). By automatically adopting new provider versions, client packages can be caught off-guard when errors are introduced (e.g., a defective provider release is published). For example, mocha is a popular package in the npm ecosystem, with more than 120K client packages and 4M weekly installations. Version 2.5.0 of mocha was released with a defect caused by a specific dependency,[1] preventing the installation of version 2.5.0 of the mocha package by any of its clients. On the same day, version 2.5.1

[1] `https://github.com/mochajs/mocha/issues/2276`

(a patch release) was released to fix the dependency issue. Another example is the defect caused by a typo introduced in version 2.4.0 of the also popular debug package (Mezzetti et al., 2018).[2] The defect affected a large number of client packages a version 2.4.1 (a patch release) was published as a same-day release to address the problem. Such examples denote the importance of same-day releases for maintaining a stable ecosystem.

The time available to revise, test, build, and document same-day releases is restricted compared to releases published within a regular schedule. When the defect on version 2.5.0 of the mocha package was identified and fixed, the build time was a concern, as expressed by a client package developer in an issue report: "*How long for build? Can one check progress somewhere?*". A time related concern could also be observed when the typo in the debug package was reported: "*Please tone it down, everybody. Mistakes happen. It's been about 30–40 minutes, and it's being worked on right now.*" As a consequence of the inherent time constraint of same-day releases, developers might need to prioritize specific activities (such as testing or building specific parts of the system). For example, Castelluccio et al. (2019) study the practice of patch uplifting by the Firefox development team. The authors observe that, occasionally, a well-established release schedule needs to be disturbed by an urgent release that contains critical fixes. Such urgent releases are "uplifted" from a development to a stable channel without being thoroughly checked. Similarly, Adams et al. (2015) show that release engineers in Firefox publish "zero-day fixes" in response to failed releases and Kerzazi and Adams (2016) study the releases of a web-based commercial system that fail "a few minutes" after being published, consequently requiring the development team to timely react to the issue. Therefore, the appropriate management of same-day releases is an important aspect to be considered by software developers.

Although prior studies examined related phenomena with same-day releases in game platforms (Lin et al., 2017) and mobile app stores (Hassan et al., 2017), the software ecosystem context was never considered. Publishing same-day releases in large software ecosystems is remarkably more impactful than publishing same-day releases for game platforms or mobile apps (as studied by prior work), where updates typically follow a push-based model. In contrast, in software ecosystems, clients use a flexible dependency specification mechanism to choose which version (or range of versions) of a provider they wish to use, which impacts on how provider releases are consumed. Hence, when provider packages quickly discover a bug or implement a feature that potentially impacts client packages in an important way (e.g., the discovered bug is known to cause crashes or the implemented feature significantly improves performance), these provider packages need to consider whether publishing a same-day release is a feasible and adequate choice. Therefore, an exploratory study of same-day releases in software ecosystems can help developers in understanding basic characteristics and consequences of publishing same-day releases in the largest software ecosystem to date (npm). By under-

[2] `https://github.com/visionmedia/debug/issues/347`

standing the prevalence (RQ1) and the types of changes that are implemented in same-day releases (RQ2), practitioners can perform an informed assessment of the relevance of managing same-day releases. In addition, an exploratory study of how client packages consume same-days (RQ3) will shade light on the impact of these releases in the ecosystem's health. Hence, in this paper, we conduct an exploratory study to uncover the fundamental characteristics and consequences of publishing same-day releases in npm. We focus on the following research questions:

Preliminary study. To understand the characteristics of the introduced changes in same-day releases, we perform a qualitative analysis over a random sample of same-day release notes. Our analysis resulted in a fine-grained set of categories that describe introduced changes in same-day releases, ultimately revealing their characteristics and scope.

Findings: We observed that the contents of the same-day release notes point to relevant changes, mainly bug fixes. The introduced changes in same-day releases can be related to the package's own business logic or to the integration with other provider packages. We also observed that introduced changes address critical issues, such as crashes, UI failures, and performance degradation.

RQ1. How often do same-day releases occur? Same-day releases imply that development activities (e.g., bug fixes, code review, and testing) were performed within a reduced time frame. Quantifying the occurrence of same-day releases will shed light on the importance of provider packages to adopt tools and procedures to help cope with such time-constrained releases.

Findings: Same-day releases represent 26% of all studied releases and are published at least once by 93% of the studied packages (i.e., packages with more than 100 clients and at least 10 releases) in npm. Same-day releases are more common for a specific set of the studied packages: 66% of all studied same-day releases are published by 25% of these packages. Also, more than one-third (39%) of the studied same-day releases are followed by another same-day release.

RQ2. How are the performed code changes in same-day releases? Same-day releases introduce important changes and are published by a large proportion of popular packages. Still, it is unknown how the source code changes required to produce same-day releases compare with other releases. In particular, we are interested in analyzing the change size (i.e., number of modified lines of code), the type of files that are changed, and the dependency updates in same-day releases compared with other releases. Therefore, in this RQ we study the code changes that are performed in same-day releases.

Findings: We observed that 32% of the same-day releases modify more lines of code than their prior releases. Also, we observed that, even with the inherent time constraint, provider packages are updated in same-day

releases. By manually investigating such provider updates, we observed that they are prevalent when client and provider packages are hosted in the same codebase.

RQ3. How do client packages adopt same-day releases? Development teams of provider packages that need to publish a same-day release can benefit from knowing how client packages typically adopt such releases (e.g., the time that it takes for clients to do so). In particular, development teams can reason about whether the practice of expediting releases is indeed beneficial to client packages.

Findings: We observed that the rate at which same-day releases are adopted by client packages is comparable with the adoption rate of regular releases. Yet, the speed at which same-day releases are adopted is significantly higher than the adoption speed of regular releases.

The aforementioned findings led us to reason about practical implications for packages that publish and consume same-day releases. We note that, despite the prevalence of same-day releases published by popular packages, client package developers have no straightforward way to determine whether a given provider release is a same-day one. We argue that third-party tools that support the automation of dependency management (e.g., Dependabot) should consider explicitly flagging same-day releases. We also note that popular package developers should carefully consider how same-day releases impact their client packages. Yet, currently there are no ecosystem-ready tools (e.g., that operate at the ecosystem-scale) to support popular provider packages in this task. When the decision to publish a same-day release is taken, developers of popular packages need to overcome the challenge of rolling out a high-quality release in a short period of time. We recommend that the pipeline for publishing a same-day release should entail: 1) taking the code of the release candidate as input, 2) running only essential tests and build steps over the performed changes, 3) deprecating the prior-to-same-day release, 4) generating release notes that clearly list the bug fixes and that explicitly flag the release as a same-day one, and 5) promptly deploying the same-day release into production.

Our study presents the following contributions:

- An empirical investigation of the purpose (preliminary study), prevalence (RQ1), code changes (RQ2), and adoption of same-day releases (RQ3) in the largest software ecosystem to date (npm);
- A discussion of practical implications for packages that publish same-day releases, as well for those that consume same-day releases;
- A supplementary material containing the set of dependencies, releases, and the difference between the contents of each pair of releases to foster future research on release and dependency management in npm.

Paper Organization. The remainder of this paper is organized as follows. Section 2 presents background material and our operational definition of same-day releases. Section 3 explains our data collection methodology. Section 4.

Section 5 presents the results of our research questions. Section 6 discusses the implications of our findings. Section 7 discusses the threats to the validity of our study. Finally, Section 8 concludes the paper.

2 Background & Operational Definitions

In this section we present background information and describe our definition of same-day releases. We initially discuss related work to release management and same-day releases (Section 2.1). Next, we describe how releases are versioned in npm (Section 2.2). Finally, we present our operational definition of same-day, prior-to-same-day, and regular releases (Section 2.3).

2.1 Release management

Release management is the set of processes that produces a final deliverable product from the source code. Software projects can adopt different activities for managing their releases (Mäntylä et al., 2015; Adams et al., 2015; Van der Hoek and Wolf, 2003; Erenkrantz, 2003) and complementary release management models were proposed in the literature (Kajko-Mattsson and Yulong, 2005; Adams and McIntosh, 2016). Figure 1 depicts a simplified release management pipeline. The simplified pipeline does not cover all possible phases and activities performed in the release management of a software project, but does explain the most common phases and activities that are relevant for the scope of our study. Essentially, our simplified pipeline synthesizes the release management models as proposed by Adams and McIntosh (2016), Lahtela and Jäntti (2011), Kajko-Mattsson and Yulong (2005), and Erenkrantz (2003). In the following, we describe each of the three phases in this pipeline.

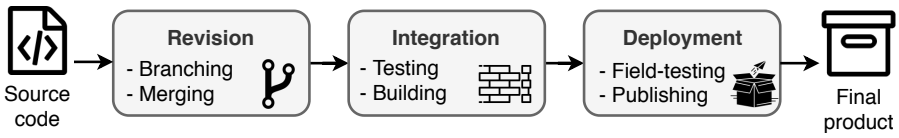

Fig. 1: Simplified release management pipeline.

Revision. In this phase, the source code that is generated by developers is committed to different active branches in the VCS and eventually merged into a production branch (the master branch). *Branching* is the process of managing different branches (or codelines) in the VCS, each one containing a version of the source code that complies with different levels of quality assurance. *Merging* is the process of combining the contents of a branch with the contents of another branch (e.g., merging a review with a master branch).

Integration. After a source code change is committed, a *build* process is started. In the build process, dependencies are linked and an executable artifact is generated. Although unit tests are likely run at development-time, automated integration processes also run a set of *tests*, including regression and integration tests.

Deployment. In the deployment phase, a release is *field-tested* and made available to the users. *Publishing* is the process of rolling out a release and making it available to the users. The published final product can be deployed and distributed to users by different means, such as package managers, web services, or mobile app stores.

Occasionally, a release needs to be expedited throughout the release management pipeline (Fox, 2002; Hamilton, 2007; Kerzazi and Adams, 2016; Hassan et al., 2017; Lin et al., 2017; Castelluccio et al., 2019). For example, Lahtela and Jäntti (2011) propose a model for release management that explicitly suggests publishing "urgent releases" when they are needed. In this scenario, developers might eventually publish a follow-up release in the same day of the previous release. Same-day releases indicate a new version that is developed and published within a restricted time-frame, regardless of the motivation behind expediting the release. The following interview excerpt illustrates the usage of same-day releases by the Mozilla development team (Adams et al., 2015):

> "*If there were a serious problem, like a huge number of crashes on a certain release (...) we would do a point release so that users wouldn't get the last release and would be automatically updated to the newer release (...) We call this a zero-day fix.*"

Kerzazi and Adams (2016) studied the "botched releases" of a commercial web app. Botched releases are defined as releases that present a malfunction shortly after ("in a few minutes") being published. The authors identified that 72 out of 320 studied releases (22.5%) are botched—a non-negligible proportion of all releases. In a study about urgent releases in the Google Play Store[3] (Hassan et al., 2017), five different patterns of urgent releases were identified, of which three have a median time to repair of one day. Urgent releases were also studied on the Steam platform[4] for games distribution. Lin et al. (2017) found that 80% of the studied games publish at least one urgent release and that the major reasons for urgent releases are feature malfunctions, crashing, and visual bugs. Lin et al. (2017) also observed that the occurrence of same-day releases in the Steam platform is associated with urgent releases.

Same-day releases are especially important in software ecosystems because packages heavily depend on each other. Prior studies show that the interdependency of npm packages makes the release of a popular package to impact many other packages that directly or indirectly depend on the former (Kikas et al., 2017; Wittern et al., 2016; Decan et al., 2017). For example, Kikas et al.

[3] `https://play.google.com/store/apps`

[4] `https://store.steampowered.com`

(2017) show that the removal of some npm packages can impact up to 30% of other packages in the ecosystem. Anecdotal evidence of this potential problem exists. For example, recently a failure in a popular package[5] with a single line of code caused a massive downtime on the npm ecosystem and a same-day release was published to solve the issue. Another well-known example is the "left-pad incident",[6] in which a package was removed from npm and led many packages in the ecosystem to fail. Within a few hours, a follow-up replacement package was published in npm and the ecosystem stability was recovered. In those two incidents, many package failures occurred because of transitive dependencies (i.e., a failure in a popular package propagated throughout the ecosystem), which denotes the risk brought by the dense interdependency of npm packages. Therefore, same-day releases of popular packages in large software ecosystems deserve proper attention. To the best of our knowledge, this is the first empirical study about same-day releases in a software ecosystem.

2.2 Versioning of releases in npm

Packages in npm adopt the semantic version numbering schema.[7] Each version number is comprised of three levels: major, minor, and patch. An example of version number is 1.2.3, where 1 is the major level, 2 is the minor, and 3 is the patch. As new versions of a package are released, any of the levels can be incremented. A *patch release* increments the patch level of the prior release (e.g., 1.2.3 to 1.2.4) and is generally associated with backward-compatible changes, such as bug fixes. In this case, we say that both releases are *adjacent releases*. Similarly, minor and major releases increment, respectively, the minor and major level of the prior release. As decimal numbers, the order of semantic version numbers is evaluated initially by the magnitude of their major level, then by their minor and patch levels (e.g., 1.1.2 comes before 1.2.0).

Each dependency in npm has an associated *versioning statement*, which specifies the version of the provider that the client package is willing to adopt. For example, a client package c that sets a versioning statement such as "P: >= 1.2.3" is specifying that the provider package p can be automatically updated to any version greater than or equal to 1.2.3. In this example, we say that any version of the provider that is greater than or equal to 1.2.3 *satisfies* the versioning statement. The actual provider version that is loaded on an installation of the client package is called the *resolved version* and depends on both the versioning statement and the existing versions of the provider.

A versioning statement can be either a *version range* (e.g., >= 1.2.3) or a *specific version* (e.g., 1.2.3). A specific version statement is satisfied by a unique

[5] `https://www.zdnet.com/article/another-one-line-npm-package-breaks-the-javascript-ecosystem/`

[6] `https://www.theregister.com/2016/03/23/npm_left_pad_chaos/`

[7] The semantic version manifest is described in `https://semver.org`. The usage of semantic version for version numbers is not mandatory in npm, but strongly recommended. See `https://docs.npmjs.com/about-semantic-versioning`

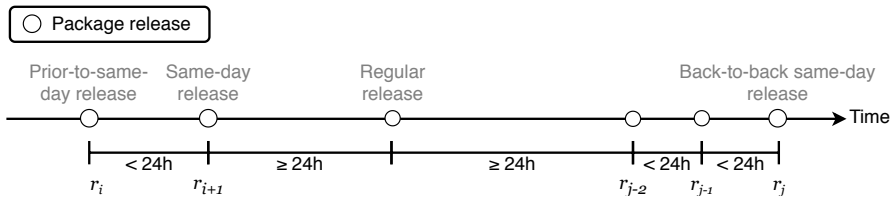

Fig. 2: The release types that are investigated in our study.

version of the provider, while the version range statement is satisfied by a range of versions of a provider. For version range statements, the resolved version is the latest provider version that satisfies the statement. When a provider releases a new version that is satisfied by the existing version range statement, an *implicit update* of this provider occurs (i.e., the provider is automatically updated). Still, an *explicit update* occurs when the client package modifies the versioning statement to update the provider.

2.3 Same-day, prior-to-same-day, and regular releases

Packages that are published in a software ecosystem have different release schedules. Some packages publish releases on a time basis (e.g., releases are periodically published based on fixed deadlines). Other packages can publish releases on a feature basis (e.g., releases are published whenever a new feature or bug fix is implemented). For this reason, it is not trivial to define what a time-constrained release is. As an effort to provide such a definition, prior research focused on releases whose time-since-previous-release is a lower outlier in the package's time-between-releases distribution (Lin et al., 2017; Hassan et al., 2017). However, this operational definition can include long-lived releases, such as releases from packages that have a large mean time-between-releases (e.g., many months). These long-lived releases do not necessarily represent time-constrained releases. Therefore, we adopt a fixed-time approach to define time-constrained releases: releases that are published within a 24-hour time window. We classify adjacent releases into three types depending on the time at which they are published, namely same-day, prior-to-same-day, or regular release. Figure 2 illustrates the three release types.

A *same-day release* is a package release that is published in npm in less than 24 hours after the previous release of the same package, according to the timezone adopted by npm to record release timestamps (UTC+0 GMT). Given two consecutive releases (in time) r_i and r_{i+1}, we say that r_{i+1} is a same-day release if the period delimited by the release timestamp of r_i and the release timestamp of r_{i+1} encompasses less than 24 hours (see Figure 2). A predecessor of a same-day release is called a *prior-to-same-day release* (which is, by definition, published in npm less than 24 hours from the following same-day release).

A same-day release that follows another same-day release is called a back-to-back same-day release. More formally, let r_{j-2}, r_{j-1}, and r_j be three consecutive releases. If the time difference between r_{j-2} and r_{j-1} is less than 24 hours and the time difference between r_{j-1} and r_j is also less than 24 hours, then r_j is a back-to-back same-day release (see Figure 2). We also observe in this formalization that, according to our definition of prior-to-same-day and same-day releases, r_{j-1} is both a same-day release (since it was published in less than 24 hours after release r_{j-2}) and a prior-to-same-day release (since it immediately precedes the same-day release r_j). Throughout this paper, when a distinction between same-day and prior-to-same-day releases is needed (e.g., when we compare metrics for same-day with the same metrics for prior-to-same-day releases), we prioritize the classification of a release as same-day. We call releases that are published after another same-day release as *back-to-back same-day releases*. Considering our previous example, the release r_3 is a back-to-back same-day release (since it was published after the same-day release r_2). Finally, releases that are not same-day nor prior-to-same-day are classified as *regular releases*.

3 Data collection

Our general goal is to understand the prevalence, code changes, and adoption of the same-day releases that are published in the npm ecosystem. The reason for choosing this ecosystem is that it is the largest and most popular software ecosystem to date.[8] Consequently, the npm ecosystem is inherently relevant in practice and provides rich data for studying same-day releases.

To accomplish our goal, we mine several pieces of data from the npm registry. Figure 3 depicts our data collection process. Four steps were employed in our data collection. The first step is performed to collect the `package.json` files from npm packages (Section 3.1). The second step selects packages for our study (Section 3.2). In the third step, we select the patch releases, identify same-day and prior-to-same-day releases, as well as identify dependencies in which the provider is updated (Section 3.3). Finally, the fourth step is to analyze how the deployed files change between adjacent npm releases (Section 3.4).

3.1 Collecting data from npm packages

In this step we collect meta-data about the packages that were published in npm between December 20, 2010 and May 13, 2019. We process the data as follows:

Obtain `package.json` files from npm packages. We fetch the `package.json` files of all published packages in the npm registry. The `package.json` file contains metadata about about all releases of a package, including the release

[8] `https://insights.stackoverflow.com/survey/2019`

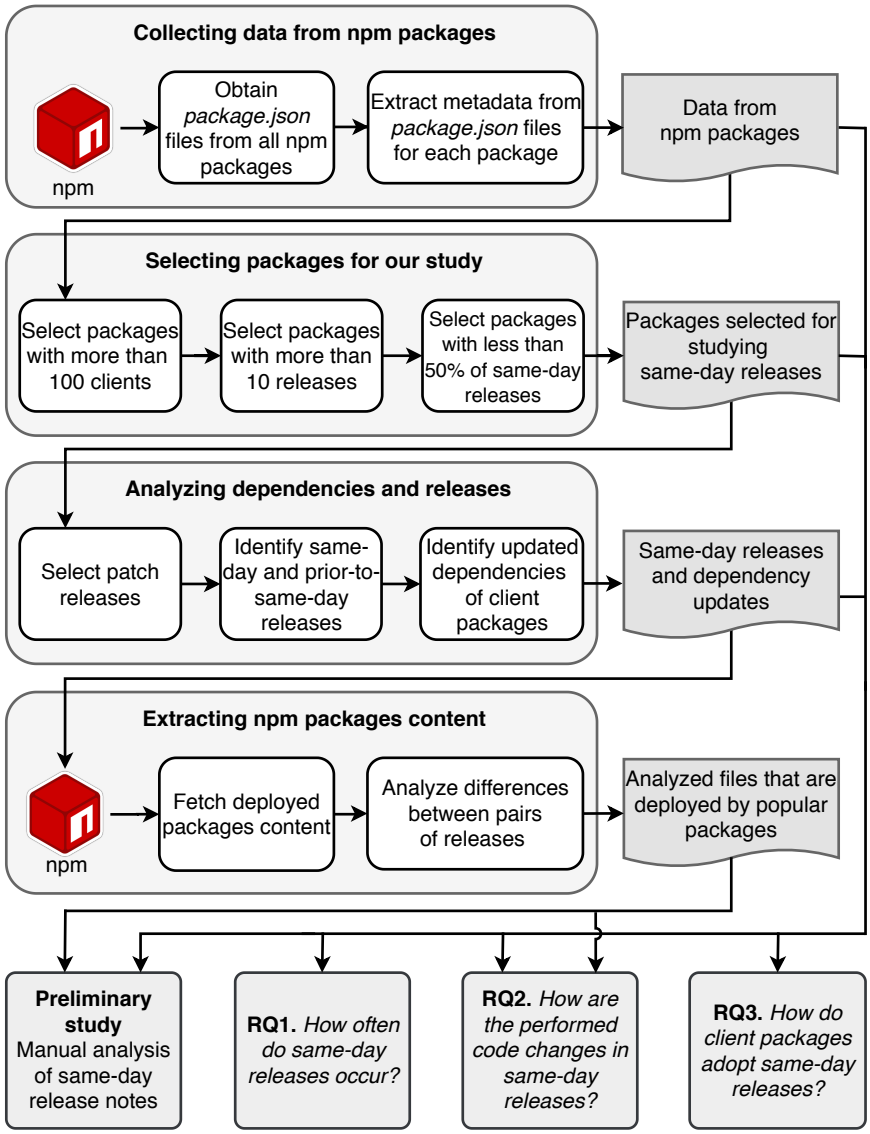

Fig. 3: Overview of our data collection process.

version number and timestamp, as well as the adopted provider packages with their respective versioning statements. We obtain the `package.json` files as a unique file that stores the history of releases and their associated metadata from the official npm replication endpoint.[9] The fetched `package.json` files are

[9] `https://replicate.npmjs.com`

limited to packages within the npm ecosystem and, therefore, do not include information about external applications that depend on an npm package.

Extract metadata from the `package.json` files of each package. We extracted the following information about the packages: the release version number, the release timestamp, the dependencies and the respective versioning statements that are used in each release. In total, we obtained the `package.json` file of 976,631 npm packages.

3.2 Selecting packages for our study

We are interested in understanding the same-day releases of popular packages in the npm ecosystem. In this step, we adopt a set of selection criteria that ensures that the phenomenon of interest is reliably captured. Our set of criteria filters out packages with a low number of client packages and releases, as well as packages that publish same-day releases too often (i.e., as part of their regular release schedule). Because our study aims at understanding same-day releases from a historical perspective, we do not apply any filter to eliminate packages that eventually become inactive (i.e., stop publishing new releases). Instead, we analyze any same-day release that was published when the selected packages were under active development. The following criteria were adopted:

Select packages with more than 100 clients. A specific goal of our research is to focus on releases that can potentially affect a large portion of the ecosystem as, for example, releases of a provider package that are adopted by many client packages. For this reason, we decide to select only releases of the popular npm packages, i.e., top-ranked packages by the number of client packages. Our definition of a popular package is based on the number of client packages that a certain provider package has *within* the ecosystem. In particular, we study packages with more than 100 clients. This threshold was selected after inspecting the distribution of the total number of clients per provider package. Other measures of popularity can be considered, in particular the download count as provided by npm and the usage of npm packages by external projects to npm (e.g., open-source projects). However, the download count data that is provided by npm might not properly reflect the importance of the same-day releases of a provider package to client packages in the ecosystem (e.g., a high number of downloads might be due to crawlers or transitive dependencies).[10] Moreover, accounting for the usage of an npm package by external projects would require defining the external projects to be sampled. Given that the population of external projects is unknown, defining such a sample and determining its statistical representativeness would be extremely challenging. Finally, the releases of external packages cannot always be easily determined or tracked, as well as which version of a provider package they are using into production at a certain given time. Therefore, we study only those packages

[10] `https://blog.npmjs.org/post/92574016600/numeric-precision-matters-how-npm-download-counts`

Table 1: Summary of the number of patch releases, number of same-day releases, number of clients, and number of providers for the popular packages.

Popular packages	Min.	Q1	Median	Mean	Q3	Max.
Number of patch releases	10	16	26	37.62	44	460
Number of same-day releases	0	3	6	9.89	12	138
Number of clients	100	158	306	2,034	928	128,999
Number of providers	1	10	18	26.29	34	200

that are popular within the ecosystem. Most importantly, understanding the prevalence, characteristics, and adoptions of same-day releases published by packages that are popular within the ecosystem is inherently relevant from a practical perspective, since these releases have a large impact potential in the entire ecosystem (e.g., a same-day release by a package p can be published to address a bug that impacts the many clients of p).

Select packages with more than 10 releases. This criterion aims at filtering out packages that rarely publish a release or that do not have a release history yet.

Select packages with less than 50% of same-day releases. We filter out packages that regularly publish same-day releases, since the development process adopted by those packages inherently deals with releases developed within a 24 hours period (i.e., same-day releases are not unusual to them). In other words, same-day releases that are published as part of a regular schedule will not reflect the effects of time constraint that we are interested in observing.

The output of this step is a set of 1,893 packages. We refer to such packages as *popular packages*. Whenever we mention a same-day release in the remainder of our study, we refer to a release of a popular package. The clients and providers of the popular packages were also included in our data set, since in our study we analyze the number of providers of the packages that publish same-day releases, as well as how the clients of such packages react to their releases. By including the clients and providers of the popular packages, a total of 504,983 packages are kept in our data set (including the popular packages). Table 1 describes statistics of the selected packages from which same-day releases are analyzed.

To characterize the selected popular packages, we extracted the developer-assigned keywords and the description of each package. A list of keywords can be assigned to a package and we extracted only the first assigned keyword, as it is likely that this is the most representative keyword for a package. During the extraction process, we removed any stop word and considered the only stem of each extracted term (i.e., we tokenized both the keywords and descriptions). Figure 4 shows the top-20 tokens from the keywords and the

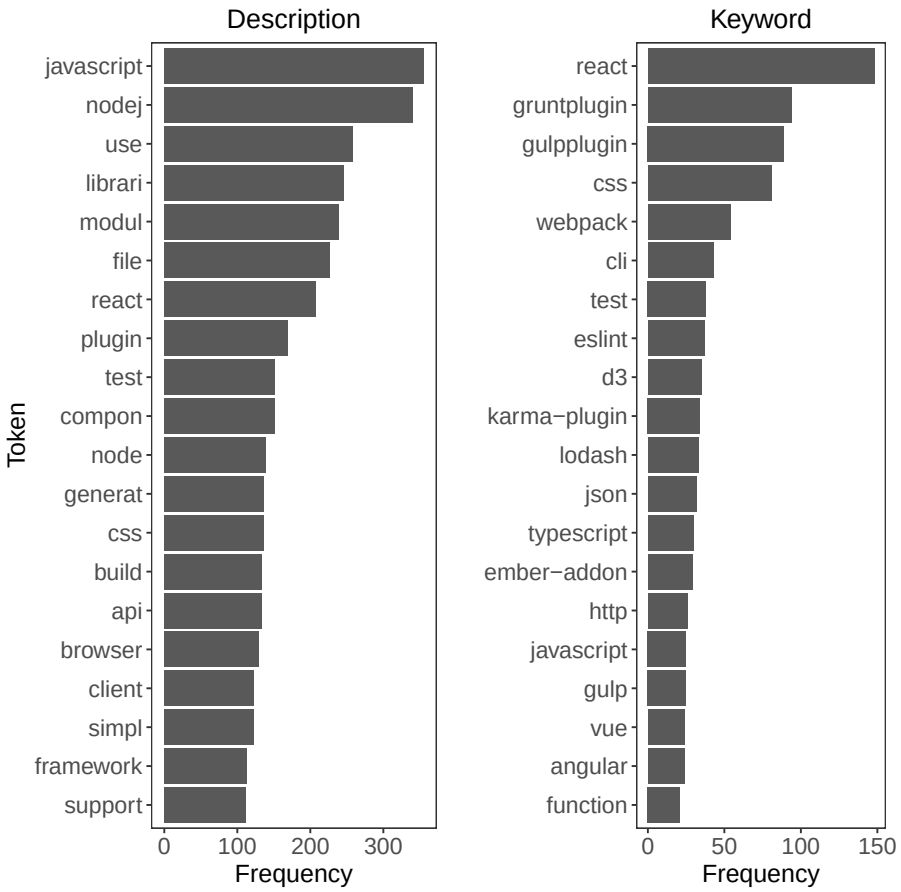

Fig. 4: The frequency of the terms that appear in the developer-assigned description and keywords of the popular packages. Stop words are removed and the terms are stemmed.

description of each popular package. Typically, such packages are described by developers as "libraries", "modules", or "components". In particular, we observe that the popular packages can be generally characterized as plugins (e.g., react, gulp, lodash, or grunt), utility functions (e.g., parsers, or json/css utilities), or frameworks (e.g., test and command-line interface).

3.3 Analyzing dependencies and releases

We analyzed the dependencies and releases of the popular packages to generate data that could be used to answer our RQs. The data was generated by the following processes:

Select patch releases. Releases in npm can be developed in parallel branches. As a consequence, major and minor same-day releases can be identified (e.g., versions 1.0.1 and 2.0.0 being released on the same-day). However, in such cases, it is clear that the same-day release (2.0.0 in the previous example) is not a patch for the 1.0.1 release. To reduce such noisy same-day releases from our analysis, we selected only the patch releases (e.g., from 1.0.0 to 1.0.1) of each studied package.

Packages in npm can publish either patch, minor, and major releases. Moreover, such releases can be interleaved, for example, when a release 1.2.3 is published as a patch to release 1.2.2 while the release 2.0.0 is already published. The occurrence of interleaved releases can misrepresent same-day releases. For example, if releases are sorted by the version number, then the releases 1.0.1 and 2.0.0 are considered adjacent releases. If those two releases are published within 24 hours, the release 2.0.0 might be considered a same-day release. However, in this case, it is clear that 2.0.0 is not patching release 1.0.1. To deal with such versioning related challenges, we select only patch releases for our study.

Identify same-day and prior-to-same-day releases. We identified all releases that were published in less than 24 hours after the prior release. Also, we identified the releases that are prior-to-same-day or regular releases (see Section 2.3).

Identify updated dependencies of client packages. For each client package release, we identified the providers that were updated in that release. To determine the updates of providers, we resolved the version of each provider according to the versioning statement used by the client package in each release and the provider versions that were released up to the date of each client release (see Section 2.2). More formally, let $d_{c,p,r}$ represent a dependency from the client package c on the provider package p in release r of c. The resolved provider version is the latest version of p that satisfies the versioning statement used by c in r. Versioning statements are parsed using our own parser,[11] which fully adheres to the npm grammar. In 22.8% of the client packages releases, at least one provider is adopted by means of a specific version (see Section 2.2)

The output of this step is a data set containing all the releases (with the identified patch same-day releases) and the dependencies of the client packages (with the identified updates) from the popular npm packages.

3.4 Extracting npm packages' contents

To understand the changes that are performed in same-day releases, we rely on data associated with the published files by npm packages. When developers publish a release in npm, they use the `npm publish` tool,[12] which packs all

[11] `https://github.com/SAILResearch/replication-npm_downgrades`

[12] `https://docs.npmjs.com/cli/publish`

public files of a package in a tarball and deploys the contents in npm. We obtain and analyze the deployed files of all releases of the popular npm packages.

Fetch deployed package contents. For each i-th release $r_{p,i}$ of a popular npm package p, we used the `npm pack` tool[13] to obtain its respective deployed tarball. This tarball contains all public files that are published in npm for a given release of a package. The rationale for analyzing the contents of the deployed release instead of analyzing the contents of the package's code base is that there is no reliable form to associate releases with branches and commits. Therefore, by analyzing the contents of the deployed package, we ensure a correct link between a release and its associated files. As a disadvantage of this approach, our analysis considers changes that are performed at a release-level, in contrast with a more fine-grained analysis of the changes at commit-level.

Analyze differences between pairs of releases. After obtaining the tarballs, we extract their contents and analyze the difference between all files of two adjacent releases $\langle r_{p,i-1}, r_{p,i} \rangle$ of a package p. This analysis calculates the number of inserted and deleted lines, as well as the contents of the added and deleted lines. As for added and removed files, we consider the contents of files that exist in release $r_{p,i}$ and do not exist on release $r_{p,i-1}$ as added lines in $r_{p,i}$. Similarly, we consider the contents of files that exist in release $r_{p,i-1}$ and do not exist on release $r_{p,i}$ as removed lines in $r_{p,i}$. Renamed files are considered as different files in $r_{p,i-1}$ and $r_{p,i}$, since we cannot identify whether a file was renamed (i.e., files are identified by their unique name). For calculating added and removed lines from the published files, we use the difflib[14] library.

The output of this step is the analysis of the differences between the deployed files of each pair of adjacent releases.

4 Preliminary study of same-day release notes contents

In this section, we present our preliminary study on the contents of the same-day release notes. The objective of our preliminary study is to better understanding the notable changes that are reported in same-day releases.

Motivation. Same-day releases are occasionally published to address issues that impact client packages in an ecosystem. Reported incidents show that the open-source community is usually able to timely react to failures in one package that end up impacting a large portion of the ecosystem (e.g., a failure in a release of a provider package that is directly and indirectly adopted by many client packages). Also, prior research shows that addressing time-sensitive issues is a common development practice (Kerzazi and Adams, 2016; Lin et al., 2017; Hassan et al., 2017; Castelluccio et al., 2019; Adams et al., 2015). However, the characteristics of the changes that are introduced in same-day releases is still an open question. For example, it is not clear whether same-day releases in fact introduce important changes (e.g., bug fixes) or are simply a result of

[13] `https://docs.npmjs.com/cli-commands/pack.html`

[14] `https://docs.python.org/3/library/difflib.html#difflib.unified_diff`

5.0.1 / 2018-02-07

...your garden-variety patch release.

Special thanks to Wallaby.js for their continued support! ❤️

🐛 Fixes

- #1838: `--delay` now works with `.only()` (@silviom)
- #3119: Plug memory leak present in v8 (@boneskull)

📖 Documentation

- #3132, #3098: Update `--glob` docs (@outsideris)
- #3212: Update Wallaby.js-related docs (@ArtemGovorov)
- #3205: Remove outdated cruft (@boneskull)

🔩 Other

- #3224: Add proper Wallaby.js config (@ArtemGovorov)
- #3230: Update copyright year (@josephlin55555)

Fig. 5: Release note example of the popular package mocha.

a routine task (e.g., merging a commit). Hence, in this preliminary study we characterize the changes that are introduced in same-day releases. Also, we investigate whether the proportion of same-day releases that have an associated release note is different from the proportion of prior-to-same-day and regular releases.

Approach. To understand the documented changes introduced in same-day releases, we perform an analysis of the same-day release notes. Release notes document the notable changes that are introduced in a new release compared to the previous release. Typically, a release note contains: 1) a list of notable changes grouped by the type of change (e.g., bug fixes, new features, documentation change, etc.), 2) the release version and the respective timestamp, and 3) links to Issue Tracking System (ITS) artifacts (such as issue reports, pull-requests, commits, or contributors). Some release notes do not include the release timestamp or any link to an ITS artifact, but all release notes have at least a list of notable changes and the respective release version. Figure 5 shows the release notes for release 5.0.1 of the popular mocha package.

As new releases of a package are published, the release notes are appended to a unique changelog file that is kept in the package's codebase repository and distributed with each release. Commonly, npm packages name their changelog files as "CHANGELOG.md". However, we also identified release notes in files that are named "HISTORY.md", "CHANGES.md", "RELEASE-NOTES.md", and "RELEASE.md", among other variations (e.g., files with HTML or txt ex-

tensions, or files with lower-case names). Release notes can also be maintained in readme files. However, to the best of our knowledge, there is no convention regarding the contents or format of release notes in readme files. Therefore, it is not trivial to automatically identify the existence of release notes in such files. For this reason, we do not consider release notes that might be occasionally stored in readme files. For further details, please refer to Section 7 (Threats to Validity). Hence, to identify changelog files of the popular npm packages, we search for files whose lower-cased name matches the following regular expression:[15]

$$(\text{change}^* \mid \text{history}^* \mid \text{release}^*).(\text{md} \mid \text{html} \mid \text{txt})$$

After identifying the changelog files, we estimate the proportion of same-day releases that have an associated release note. To establish a baseline, we also calculate the proportion of prior-to-same-day and regular releases that have an associated release note. We used the Fisher's Exact Test ($\alpha = 0.05$) with Holm-Bonferroni correction to determine whether there is any relation between the type of release (same-day, prior-to-same-day, and regular) and the publication of a release note. More formally, let SDR_p, PSDR_p, and REG_p be, respectively, the set of same-day, prior-to-same-day, and regular releases of a package p. We tested two null hypotheses: $H_0 : P_{\text{RN}}(\text{SDR}_p) = P_{\text{RN}}(\text{PSDR}_p), \forall p \in P$ and $H_0 : P_{\text{RN}}(\text{SDR}_p) = P_{\text{RN}}(\text{REG}_p), \forall p \in P$, where P_{RN} is the proportion of releases that have a release note. The magnitude of the differences is calculated using the Cramér's V estimator of effect size, which is interpreted according to the following thresholds (Louis M. Rea, 2014)

$$\text{Effect size} = \begin{cases} \text{negligible}, & \text{if } 0 \leq |V| < 0.1 \\ \text{weak}, & \text{if } 0.1 \leq |V| < 0.2 \\ \text{moderate}, & \text{if } 0.2 \leq |V| < 0.4 \\ \text{relatively strong}, & \text{if } 0.4 \leq |V| < 0.6 \\ \text{strong}, & \text{if } 0.6 \leq |V| < 0.8 \\ \text{very strong}, & \text{if } 0.8 \leq |V| \leq 1 \end{cases}$$

Next, we proceed with a manual analysis of the same-day release notes (i.e., the release notes of same-day releases). We thoroughly read the same-day release notes' contents as well as the ITS artifacts that are linked in the release notes. During our reading process, we seek for deriving a detailed description of the notable changes of same-day releases. In particular, we carry out an open coding process (Stol et al., 2016), resulting in a set of categories that describe the typical changes that are introduced in same-day releases. During our open coding process, we achieved saturation after analyzing a random sample of 104 same-day release notes. Afterwards, we compare our categorization with a pre-existent categorization schema as proposed by Moreno et al. (2014). Therefore,

[15] In our regular expression notation, the | operator is a logical OR and the * operator is a non-greed zero-or-more matcher of any character. All other symbols are literal characters.

Table 2: Statistical significance of the difference between the proportion of same-day, prior-to-same-day, and regular releases that have an associated release note.

Comparison	**Corrected p-value**	**Cramér's V**
Same-day vs. prior-to-same-day	2.3×10^{-4}	0.021 (negligible)
Same-day vs. regular releases	2.2×10^{-16}	0.041 (negligible)

in a next step, we also categorize the 104 same-day release notes with Moreno's categorization schema. The categories derived from our open coding process differs from the categories as proposed by Moreno et al. (2014) with respect to the level of detail. For example, the content of a release note can be categorized as a bug fix according to the categorization schema as proposed by Moreno et al. (2014), while our categorization denotes the type of change employed to fix the bug (e.g., changing the business logic) or, even, the type of bug that drove the change (e.g., fixing a memory leak).

We use an infographic (Lex et al., 2014) to visualize the occurrence and co-occurrence frequencies of the categories found in same-day release notes. In this infographic, the axis labelled "occurrence frequency" represents the total number of same-day release notes whose content was categorized with a certain category. In turn, the axis labelled "co-occurrence frequency" represents the number of same-day release notes of which the content was categorized into one, two, or three categories at once. The co-occurrent categories are identified by a dot matrix shown in the inforgraphic. A column-wise filled dots in the matrix represent the categories that co-occur in a same-day release note (exclusive categories appear as a single filled dot).

Result 1: ***32% of the same-day releases have an associated release note.*** In turn, 34% of the prior-to-same-day releases and 37% of the regular releases have a release note. Table 2 shows that, although the difference between the proportion of same-day and prior-to-same-day releases that have a release note is statically significant, the effect sizes are negligible. The result of the comparison between same-day and regular releases is analogous. Hence, we conclude that the existence of release notes, in practice, does not seem to be related to the release type. Nevertheless, the proportion of releases that have a release note is relatively low (approximately one third of all releases). The lack of release notes can be problematic for client packages, since they become unable to know which changes were introduced in a release and, therefore, cannot assess the impact of the introduced changes.

Result 2: ***The analyzed same-day release notes typically refer to a fixed bug.*** In Figure 6, we summarize the categorization of the studied release notes according to the categorization schema as proposed by Moreno et al. (2014). The category "fixed bugs" occurred in more than 60 same-day release notes. Also, the content of 45 same-day release notes was categorized exclusively as "fixed bugs" (see the first column of the dot matrix), whereas

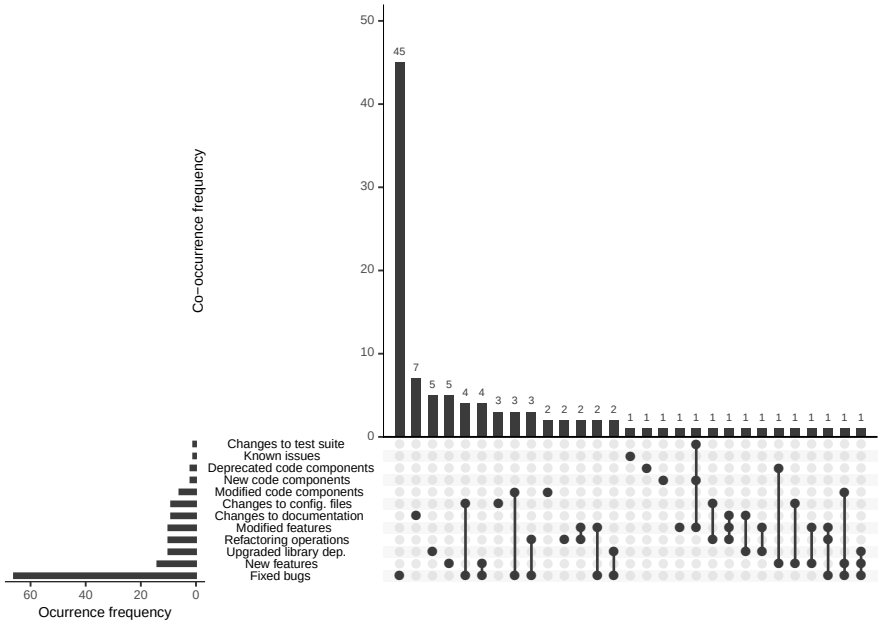

Fig. 6: Summary of the categorization of notable changes in 104 same-day releases.

the content of 4 same-day release notes was categorized with both "fixed bugs" and "changes to configuration files" categories (see the fifth column of the dot matrix). Despite the plurality of categories and co-occurrence of categories shown in Figure 6, we can observe that fixed bugs are more prevalent than any other category.

Result 3: ***The most common changes in the studied same-day releases are to change business logic, to correct the integration of a provider package, or to avoid an unrecoverable failure.*** In Table 10 (Appendix A), we describe in details the 28 categories of changes that were introduced in same-day releases. While "changes in business logic" suggest that many changes introduced in same-day releases rely on the behaviour of the popular package itself (i.e., the package that publishes the same-day release), "correctly integrate provider" suggests that the changes introduced in same-day releases rely on the interaction between the popular package and its providers. Also, "avoid crashing" and "fix UI error" suggest that the changes introduced in same-day releases solve serious issues that affect client packages.

During our manual analysis, we observed that even routine tasks such as those categorized as "VCS chores" or "Fix typo" can address relevant issues. For example, release 0.4.20 of the config package was published to address an issue caused by a non-merged commit. The following dialogue was captured in issue report #47 of the config package:

– "*Unfortunately this was the last commit using the old philosophy (commit to version branches vs. master), and it got missed.*"
– "*(...) Fix of _ diffDeep function was critical for me.*"

Another issue report exemplifies simple yet serious issues that are addressed in same-day releases. A failure caused by an improperly placed comment in a CSS file was fixed in a same-day release:

– "*This CSS: (...) Produces this error (...) This is kicked because of the comment.*" (Issue report #163 of package postcss-import@8.0.2)

We also observed a same-day release note categorized as "Improve documentation" that admits the introduction of technical debt:

"*Ideally the plugin would check for permissions and raise a more helpful error, but a note in the readme should help until then*" (Issue report #13037 of package gatsby@2.3.6)

Preliminary study

We performed a manual analysis of the release notes of same-day releases. We observed that:

• There is a negligible difference between the proportion of same-day, prior-to-same-day, and regular releases that have a release note. Roughly only one third of the releases are accompanied by a release note.

• In general, release notes are poorly described, requiring examination of additional documentation to clarify the changes introduced in a release.

• An analysis of the release notes and additional documentation of same-day releases shows that these releases typically refer to a bug fix.

• The typical changes in same-day releases include fixing the package's own business logic, integrating provider packages, and fixing unrecoverable failures.

5 Research Questions

In this section, we present the motivation, approach, and results for each of our RQs.

5.1 RQ1: How often do same-day releases occur?

Motivation. The inherent time constraint of same-day releases can compel developers to expedite typical release activities. However, the extent to which

development teams of packages in npm need to publish same-day releases is not understood. Quantifying the phenomenon of same-day releases is a first step towards determining how important it is for package development teams to adopt a suitable strategy to manage such releases.

Approach. To investigate how often same-day releases are published by popular npm packages, we first calculate the proportion of all releases that are same-day releases and the proportion of popular packages that publish at least one same-day release. More formally, let PP be the set of all popular packages, SDR_p the set of same-day releases of a package p, and R_p the set of all releases of a package p. The proportion of releases that are same-day, P_{SD}, is given by:

$$P_{\mathrm{SD}} = \frac{\sum_{p \in \mathrm{PP}} |\mathrm{SDR}_p|}{\sum_{p \in \mathrm{PP}} |\mathrm{R}_p|}$$

In turn, the proportion of all popular packages that publish at least one same-day release, P_{1SD}, is given by:

$$P_{\mathrm{1SD}} = \frac{|\{p : |\mathrm{SDR}_p| > 0\}|}{|\mathrm{PP}|}, \forall p \in \mathrm{PP}$$

We also investigate how often popular packages have their next regular release schedule interrupted by a same-day release. To do so, we calculate the interruption ratio of each popular package. We define the interruption rate of a package as the proportion of regularly scheduled releases that are interrupted by a same-day release or a sequence of same-day releases (see Figure 7). To formally define the interruption rate, let SDR_p, BSDR_p, PSDR_p, and REG_p be, respectively, the set of same-day, back-to-back same-day, prior-to-same-day, and regular releases of a package p. The interruption rate IR_p of a package p is defined as the number of same-day releases, excluded back-to-back same-day releases, divided by the number of regular and prior-to-same-day releases of that package:

$$\mathrm{IR}_p = \frac{|\mathrm{SDR}_p \setminus \mathrm{BSDR}_p|}{|\mathrm{REG}_p \cup \mathrm{PSDR}_p|}.$$

We also investigate whether some popular packages publish more same-day releases than others. To this end, we analyze a cumulative histogram showing the proportion of packages that publish a given proportion of same-day releases. We also calculate the proportion of back-to-back same-day releases (i.e., same-day releases followed by another), as well as the proportion of packages that publish back-to-back same-day releases at least once.

Finally, we check whether there is an association between the number of same-day releases and the number of overall releases of the popular packages.

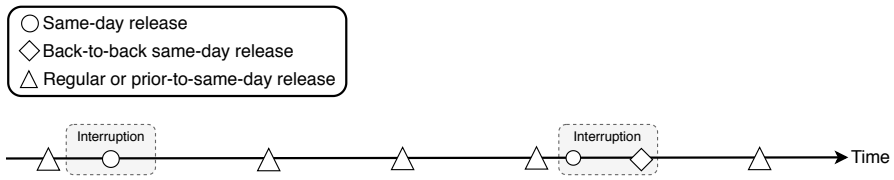

Fig. 7: The interruption of a regularly scheduled release by a same-day release.

Table 3: The distribution of the interruption rate for popular packages.

	Min.	**Q1**	**Median**	**Mean**	**Q3**	**Max.**
Interruption rate	0.0%	13.1%	22.2%	22.2%	30%	85.7%

We calculate the Spearman's ρ measure of correlation between those two variables. The choice of a rank-based measure of correlation is due to the skewness and the presence of outliers in both variables. Also, we interpret the correlation measure according to the following classification (Schober et al., 2018):

$$\text{Correlation} = \begin{cases} \text{negligible}, & \text{if } 0 \leq |\rho| \leq 0.1 \\ \text{weak}, & \text{if } 0.1 < |\rho| \leq 0.39 \\ \text{moderate}, & \text{if } 0.39 < |\rho| \leq 0.69 \\ \text{strong}, & \text{if } 0.69 < |\rho| \leq 0.89 \\ \text{very strong}, & \text{if } 0.89 < |\rho| \leq 1 \end{cases}$$

Result 4: ***The vast majority of the popular packages (96%) publish at least one same-day release.*** Also, 26% of the releases of a popular package are followed by a same-day release (i.e., one same-day release for every four regular releases in average), which suggests that activities aimed at managing such expedited releases can be beneficial to popular packages. Also, the median interruption rate of the popular packages (see Table 3) is 22%, which means that generally speaking, for approximately each five regularly scheduled releases of a popular package, one is interrupted by a same-day release.

Result 5: ***Most of the studied same-day releases are published by a specific subset of the popular packages.*** Figure 8 depicts the proportion of packages that publish a given proportion of same-day releases. We can observe that the number of same-day releases per package is fairly skewed. In particular, 25% of all studied popular packages publish 65% of all studied same-day releases. This result shows that there are packages that are more prone to publish same-day releases. For such packages, the implementation of practices to manage same-day releases might be fundamentally important. Also, the clients of this particular set of popular packages are more susceptible to adopt a same-day release.

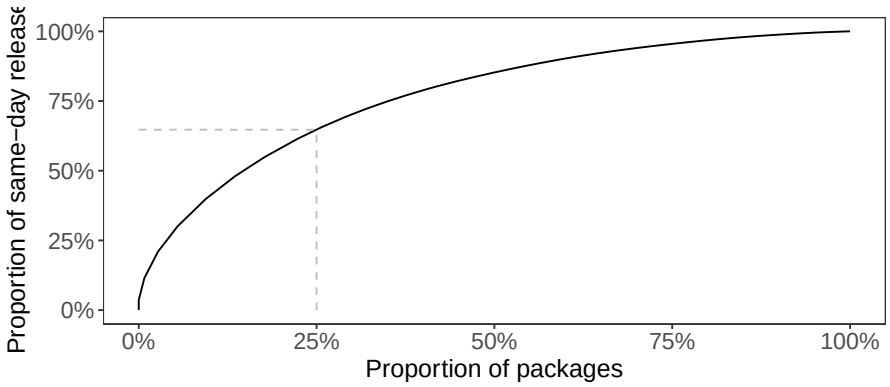

Fig. 8: Cumulative histogram for the proportion of published same-day releases per package.

Result 6: ***Same-day releases are commonly followed (39%) by another same-day release.*** That is, in approximate terms, for every three same-day releases, two of them are consecutive in time and less than 24 hours apart (on average). Also, 70% of the popular packages publish at least one back-to-back same-day release. This observation shows that, once a same-day release is published, it is not a rare scenario in which another same-day release is published. Also, a small set of the popular packages (25%) publish a large proportion of the back-to-back same-day releases (76%).

Result 7: ***The number of same-day releases and the number of releases of the popular packages are strongly correlated.*** For the majority of the popular packages, as the number of releases of a package increases, the number of same-day releases also increases. The relation between the number of releases and the number of same-day releases is shown in Figure 9. There is a strong monotonic relation between the number of releases and the number of same-day releases of a package (Spearman's $\rho = 0.73$, p-value < 0.05). The monotonic relation between the number of releases and the number of back-to-back same-day releases is also strong (Spearman's $\rho = 0.85$, p-value < 0.05).

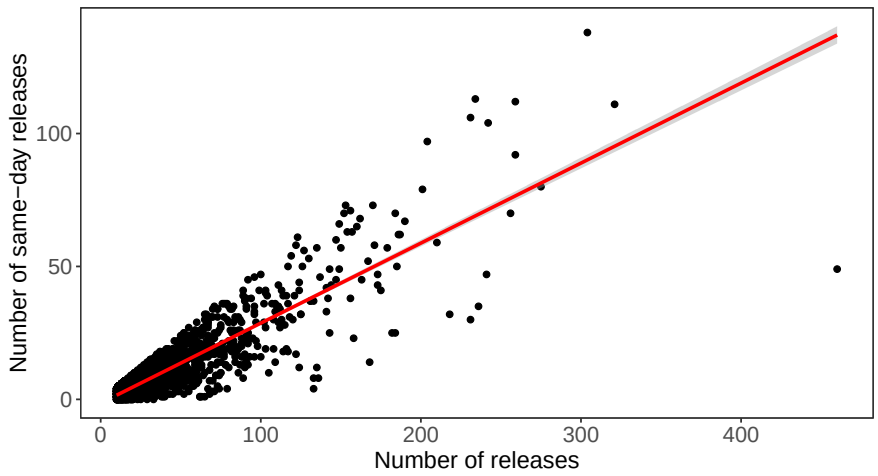

Fig. 9: The relation between the number of same-day and the number of total releases of each popular package.

How often do same-day releases occur?

We analyzed the proportion of same-day and of popular packages that publish at least one same-day release. We also analyzed how often a same-day releases is followed by another. We observed that:

- Same-day releases are more than one-quarter (26%) of all releases of a popular package.
- 25% of all popular packages publish 65% of all same-day releases.
- More than one-third (39%) of the same-day releases are followed by another same-day release.

5.2 RQ2: How are the performed code changes in same-day releases?

In this section, we describe the differences between the code changes performed in same-day and other release types. We focus on two main aspects of the code changes. In Section 5.2.1, we study a set of *change metrics* for same-day releases. In Section 5.2.2, we study the *modified file types* in same-day releases. We adopt the performed changes in other release types (prior-to-same-day and regular releases) as a baseline for studying the performed changes in same-day releases.

5.2.1 Change metrics

Motivation. Change metrics relate to the changes that are performed across different states of a software system. These metrics are useful to measure how much maintenance activity is performed during a revision or a release of a software. The analysis of change metrics can reveal whether certain types of changes are performed in same-day releases, as well as the extension of the undergone maintenance in such releases, compared with other release types. Prior research also shows that change metrics are useful to assess software quality (Graves et al., 2000; Nagappan and Ball, 2005; Moser et al., 2008) and maintainability (Eski and Buzluca, 2011).

Approach. We calculate and compare a set of change metrics for same-day and other release types. The following metrics were calculated for all popular package releases:

Number of modified lines of code (mLOC): We calculate how many lines of code are modified (inserted or deleted) in the files that are published in npm as part of a package's release (see Section 3.4 for a description of how such files are obtained). More formally, for each i-th release $r_{p,i}$ of a package p, $\text{mLOC}_{r_{p,i}}$ is the number of modified lines of code in the files associated with $r_{p,i}$ (in comparison with the previous release $r_{p,i-1}$).

Number of modified source code files: The analysis of the number of modified source code files in a release reveals how much change occurs in source code files instead of other files related to, for example, configuration or documentation. Hence, this metric measures the extent to which a change introduces modifications in the behaviour of a package. The behaviour of an npm package is embedded into JavaScript source code files. However, two other popular languages in npm that are compiled to JavaScript (and, therefore also embed package's behaviour) are TypeScript and CoffeeScript. We identify source code files by using a regular expression that captures file names with the extension "`.js`" (for JavaScript), "`.ts`" (for TypeScript), or "`.coffee`" (for CoffeeScript) within a "test", "src", "lib" or root directory.

Number of updated providers: Updating a provider package requires a migration effort that, in many cases, involves understanding the implications of the introduced changes in the new provider version. Understanding such implications can require time from developers, which could be scarce for same-day releases. Thus, this metric can help understand the extent to which developers are dedicating their effort to update providers in a release. To calculate this metric, we sum the number of provider packages that are updated in a release. To detect whether a provider package p is updated in a release $r_{c,i}$ of a client package c, we resolve the version of p that is used by c in the releases $r_{c,i-1}$ and $r_{c,i}$. If the resolved version of p in $r_{c,i-1}$ is smaller than the resolved version of p in $r_{c,i}$, then an update of p is accounted for in $r_{c,i}$.

We compare the value for the aforementioned metrics between different release types of the popular packages. More formally, let PP be the set of all

popular packages. Also, let SDR_p, PSDR_p, and REG_p be, respectively, the set of same-day, prior-to-same-day, and regular releases of a package p. Now let $m(r_{p,i})$ denote the calculation of a metric m at the i-th release $r_{p,i}$ of a package p. To compare same-day releases with prior-to-same-day releases for a metric m, we use the Wilcoxon Signed-rank ($\alpha = 0.05$) (Bauer, 1972) test (i.e., a non-parametric test for paired samples) to evaluate the following null hypothesis:

$$H_0 : m(r_{p,i}) = m(r_{p,i-1}), \forall i, p : r_{p,i} \in \text{SDR}_p \wedge r_{p,i-1} \in \text{PSDR}_p \wedge p \in \text{PP}$$

In addition, to compare same-day releases with regular releases for a metric m, we once again apply the Wilcoxon Signed-rank ($\alpha = 0.05$) test and evaluate the following null hypothesis:

$$H_0 : \mathit{Median}(m(r_{p,i})) = \mathit{Median}(m(r_{p,j})), \forall i, j, p : r_{p,i} \in \text{SDR}_p \wedge r_{p,j} \in \text{REG}_p \wedge p \in \text{PP}$$

The calculated p-values are corrected with the Holm method for multiple comparisons (same-day vs. prior-to-same-day and regular releases). The magnitude of the difference between the distributions is assessed using the Cliff's Delta d estimator of effect size (Cliff, 1996). We interpret the value of d according to the following thresholds (Romano et al., 2006):

$$\text{Effect size} = \begin{cases} \text{negligible}, & \text{if } 0 \leq |d| \leq 0.147 \\ \text{small}, & \text{if } 0.147 < |d| \leq 0.33 \\ \text{medium}, & \text{if } 0.33 < |d| \leq 0.474 \\ \text{large}, & \text{if } 0.474 < |d| \leq 1 \end{cases}$$

For each metric m whose difference across release types yields a non-negligible effect size, we visualize the data distribution of two ratios. These ratios are the *same-day to prior-to-same-day for a metric m* and the *same-day to regular for the median of a metric m*. We define them as follows:

Same-day to prior-to-same-day ratio for a metric m: The same-day to prior-to-same-day ratio $SP(r_{p,i}, m)$ for a metric m at a same-day release $r_{p,i}$ with $p \in \text{PP}$ is:

$$SP(r_{p,i}, m) = \frac{m(r_{p,i})}{m(r_{p,i-1})}, \text{where } r_{p,i} \in \text{SDR}_p \wedge r_{p,i-1} \in \text{PSDR}_p$$

Same-day to regular ratio for the median of a metric m: The same-day to regular ratio $SR(p, m)$ for the metric m of a package $p \in \text{PP}$ is:

$$SR(p, m) = \frac{\mathit{Median}(m(r_{p,i}))}{\mathit{Median}(m(r_{p,j}))}, \forall i, j : r_{p,i} \in \text{SDR}_p \wedge r_{p,j} \in \text{REG}_p$$

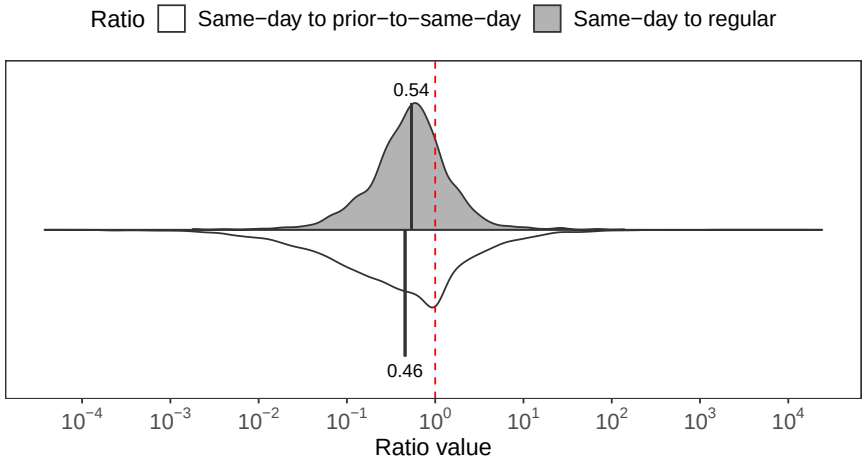

Fig. 10: Same-day to prior-to-same-day and same-day to regular ratios for the mLOC metric. The red dotted line indicates a ratio of 1.

Result 8: ***In 32% of the same-day releases, the number of modified lines of code is larger than in their associated prior-to-same-day releases.*** In Table 4 we show descriptive statistics for mLOC per release type. Analysis of this table reveals that the median mLOC for same-day releases is lower than that for prior-to-same-day. Such a result is unsurprising given the time-constrained nature of same-day releases. Indeed, as shown in Table 5, the difference between the two distributions is statistically significant with a non-negligible effect size. The results that stem from comparing mLOC between same-day releases and regular releases are analogous (with the exception of a slightly higher absolute value of effect size), as shown in Table 6. Nevertheless, we do observe the existence of same-day releases with an unusual mLOC value. In particular, we note that 32% of the same-day releases have a higher mLOC than their associated prior-to-same-day release (SP ratio higher than one). This result can be seen in Figure 10, which depicts the SP ratio for the mLOC metric (white distribution). An analysis of the figure reveals that the ratio can be even larger than 10 in some cases. Similar observations can be seen for the SR ratio (grey distribution in Figure 10). For instance, we note that 24% of the popular packages have a higher median mLOC in same-day releases compared to regular releases. We thus conclude that, albeit not the norm, same-day releases can have large mLOC. Larger release change sizes pose an inherent trade-off to same-day releases. On the one hand, large changes are a sign that the new release has undergone significant maintenance activity, including the introduction of bug fixes and new features (see Section 4). On the other hand, large changes are susceptible to the introduction of new bugs, especially when developed in a constrained time interval.

Table 4: Summary of the mLOC metric for same-day, prior-to-same-day, and regular releases.

Release type	Min.	Q1	Median	Mean	Q3	Max.
Same-day	2.0	6.0	14.0	243.2	42.0	178,314.0
Prior-to-same-day	1.0	11.0	34.0	544.4	126.0	483,987.0
Regular	2.0	11.0	32.0	602.1	117.0	679,894.0

Table 5: Statistical difference between the change metrics for same-day releases vs. prior-to-same-day releases (p-values are adjusted).

Metric	Significant?	Effect size
Number of modified lines of code	Yes ($p = 2.2 \times 10^{-16}$)	Small ($d = -0.323$)
Number of modified source code files	Yes ($p = 2.2 \times 10^{-16}$)	Negligible ($d = -0.142$)
Number of updated providers	Yes ($p = 2.2 \times 10^{-16}$)	Large ($d = -0.521$)

Table 6: Statistical difference between the change metrics for same-day releases vs. regular releases (p-values are adjusted).

Metric	Significant?	Effect size
Median of modified lines of code	Yes ($p = 2.2 \times 10^{-16}$)	Medium ($d = -0.366$)
Median of modified source code files	Yes ($p = 2.2 \times 10^{-16}$)	Negligible ($d = -0.138$)
Median number of updated providers	Yes ($p = 2.2 \times 10^{-16}$)	Large ($d = -0.749$)

Result 9: ***Despite the time-constrained nature of same-day releases, 17% of them still update at least one provider.*** In turn, 62% of the prior-to-same-day releases have at least one provider updated. The difference between the distributions of the number of provider updates in same-day releases and the other release types is statistically significant and the effect size is large (see Tables 5 and 6). The distributions of the same-day to prior-to-same-day ratio and the same-day to regular ratio for the number of updated providers are shown in Figure 11. Also, we observe that 18% of the providers that were updated in the prior-to-same-day release are the same as the providers that were updated in the same-day release. By manually analyzing such cases, we found that they are predominant in packages that are maintained in monolithic repositories (Jaspan et al., 2018) (i.e., a single repository that hosts multiple packages), as for example the popular angular[16], babel[17], and react[18] packages. Such packages are part of a main project that maintains several independent packages in the same repository. Each independent package is used as dependency by other packages in the main project (in turn, such packages can be

[16] `https://github.com/angular/angular/tree/master/modules`
[17] `https://github.com/babel/babel/tree/master/packages`
[18] `https://github.com/facebook/react/tree/master/packages`

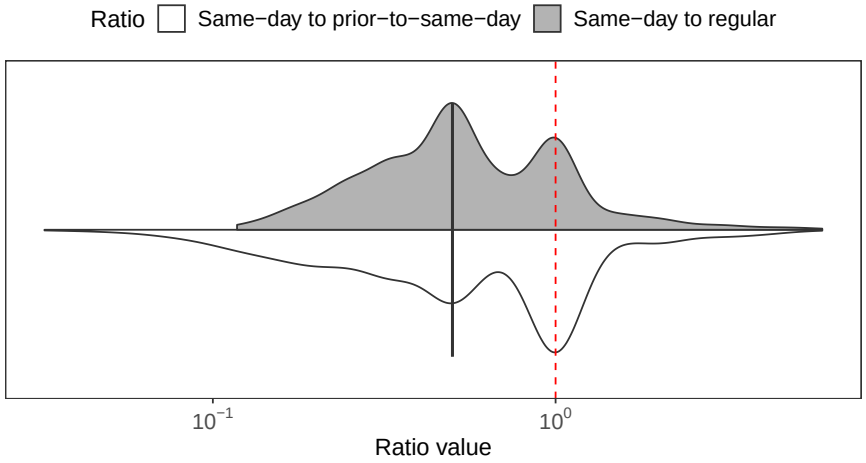

Fig. 11: Same-day to prior-to-same-day and same-day to regular ratios for the number of updated providers. The red dotted line indicates a ratio of 1.

reused by other packages in npm). This result suggests that, although packages pertaining to the same main project are independently maintained, there is an association between the releases of such packages that can drive same-day releases.

5.2.2 Modified file type

Motivation. In Section 5.2.1, we analyzed how the values of change metrics from same-day releases compare those to those from prior-to-same-day and regular releases. However, changes can be performed in different portions of a software package. For example, releases that modify the source code (e.g., files that control the package's business logic) are likely to be either more important or impactful to client packages than those releases that modify documentation-related files. In this RQ, we want to determine whether the time constraint of same-day releases has any effect on the locations of changes. By answering this RQ, we provide complementary evidence regarding the source code changes that are performed in same-day releases, beyond the scope of the change metric measurement performed in Section 5.2.1.

Approach. Specific file types are stored in specific directories within the directory structure of npm packages. For example, a folder named `test` is likely to store test-related files. We studied the structure of the directories that store the modified files in same-day and other release types. Figure 12 depicts our approach:

1. *Determine files that are modified by each package:* To determine which files are modified in each release of a popular package, we rely on the files

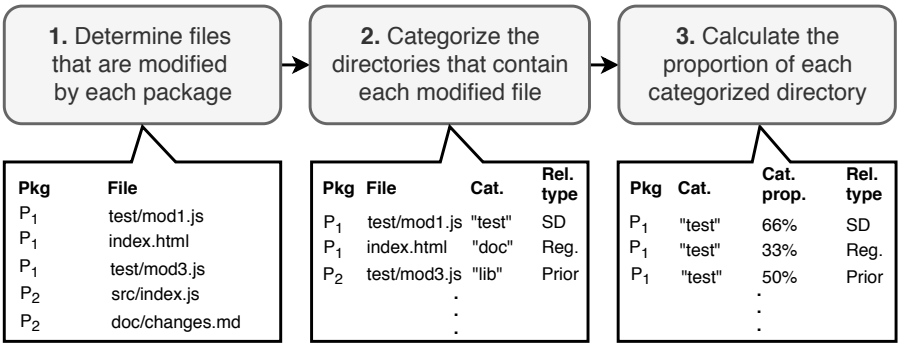

Fig. 12: Our approach to study the modified file types in same-day and other release types.

that are published with each package release (see Section 3.4). For each published file of a popular package p, we calculate the differences (per line) in the file contents between release $r_{p,i-1}$ and release $r_{p,i}$ (considering that release $r_{p,i-1}$ is adjacent to release $r_{p,i}$). We consider that a file is modified between two adjacent releases whether the difference of the content is non-empty.

2. *Categorize the directories that contain each modified file:* To categorize the directories that contain each modified file, we initially manually inspect the directory structure of a random sample of the popular npm packages. As a result of this manual inspection, we describe a set of directory types that are typically found in the directory structure of popular npm packages. Each directory type clearly separates files by their role (e.g., test-related files are stored in a different directory than documentation-related files). We also derive a set of regular expressions that are able to identify each directory type within the directory structure of a package. After identifying each directory type, we performed a sanity check to determine the accuracy of our method to categorize the packages' directories. In our sanity check, we verify whether the directories were correctly categorized according to the regular expressions. We observed that our regular expressions are able to accurately identify the directory types. For the sake of simplicity, we refer to files that are contained in a certain directory category as related files to that category. For example, we refer to files that are contained in a directory categorized as "test" as "test-related files". We used regular expressions to categorize the directory structure according to the following categories:

test Directory containing test-related files. Any directory whose name contains the string "test", "tests", "spec", or "specs" is deemed as a test directory.

lib Directory containing library-related files. Any directory whose name contains the string "lib" or "libs" is deemed as a library directory.

src Directory containing production-related files. Any directory whose name contains the string "src" is deemed as a production directory.

doc Directory containing package's documentation. Any directory whose name contains the strings "doc*" (such as "doc", "docs" or "documentation'), "example", or "examples" is deemed as a documentation directory.

build Directory containing build-related files. Any directory whose name contains the string "build" or "ci" (for "continuous integration") is deemed as a build directory.

others Directories that do not fit in the prior categories, or contain files from a wide variety of file types, including the root directory.

3. *Calculate the proportion of each categorized directory:* For each release type, we calculate the proportion of releases that modify files that are stored in each directory category. More formally, we define $P_{d_c}(\text{SDR}_p)$, $P_{d_c}(\text{PSDR}_p)$, and $P_{d_c}(\text{REG}_p)$ as the proportion of same-day, prior-to-same-day, and regular releases of a package p that modifies a file in a directory categorized as d_c, respectively. For example, a package p_1 can modify test-related files in 66% of its same-day releases. Similarly, p_1 can modify test-related files in 33% and 50% of its regular and prior-to-same-day releases, respectively. In this case, $P_{\text{"test"}}(\text{SDR}_{p_1}) = 0.66$, $P_{\text{"test"}}(\text{PSDR}_{p_1}) = 0.33$, and $P_{\text{"test"}}(\text{REG}_{p_1}) = 0.5$.

We use the Wilcoxon Signed Rank ($\alpha = 0.05$) to test the null hypothesis that the proportion of same-day releases that modify files in a certain directory category is equal to the proportion in prior-to-same-day and regular releases. The comparisons were paired per package, i.e., the proportion of same-day releases that modify a certain file type is compared with the proportion of prior-to-same-day and regular releases of the same package that modify that file type. For example, for the directory category "test", we test the null hypotheses $H_0 : P_{\text{"test"}}(\text{SDR}_p) = P_{\text{"test"}}(\text{PSDR}_p)$, where $P_{\text{"test"}}(\text{SDR}_p)$ and $P_{\text{"test"}}(\text{PSDR}_p)$ are, respectively, the proportion of test-related files that are modified in SDR_p (the set of all same-day releases of a package p) and PSDR_p (the set of all prior-to-same-day releases of a package p). We use the Holm correction method to correct the obtained p-values for multiple comparisons (across combinations of directory categories and release types).

Result 10: ***There is no practical difference between the proportion of modified file types between same-day releases and other release types.*** Despite the difference in change size between same-day releases and other release types (see Section 5.2.1), the type of files that are modified are essentially the same regardless of the release type. In particular, we observe that, for all directory categories, with the exception of the "doc" directory for same-day vs. prior-to-same-day releases (p-value = 0.334, Holm corrected), the differences are all statistically significant. However, for the statistically significant differences, the effect size is negligible. Figure 13 shows the distribution of the ratio between the proportion of modified file types in same-day and prior-to-same-day releases, as well as between same-day and regular releases of the same package. This observation reinforces the relevance of same-day releases

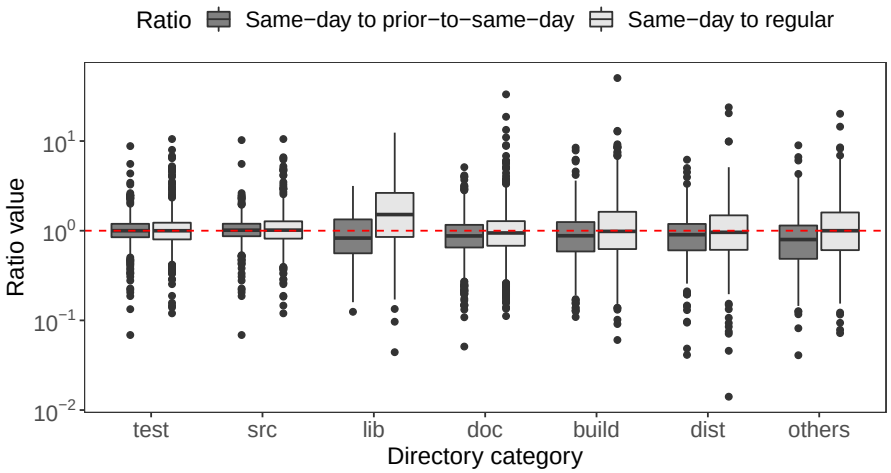

Fig. 13: The ratio between the proportion of modified file types between same-day and prior-to-same-day releases, as well as between same-day and regular releases.

since, in general, they modify the same amount of each file type compared with prior-to-same-day and regular releases.

How are the performed code changes in same-day releases?

We analyzed the code change metrics and the modified file types of same-day releases. We observed that:

- 32% of the same-day releases modify more lines of code than their associated prior-to-same-day releases.
- 17% of the same-day releases update at least one provider. Such updates are predominant for provider packages that are maintained in monolithic repository as the client package.
- There is no practical difference in the proportion of modified file types between same-day releases and other release types.

5.3 RQ3: How do client packages adopt same-day releases?

Motivation. In a software ecosystem environment, same-day releases have an important role, since problematic releases of a popular package can cause issues in a large proportion of its client packages. Development teams that need to publish a same-day release can benefit from knowing how client packages adopt such releases. Therefore, same-day releases can target the right set of client

packages and the adoption of such releases can be maximized. In fact, a prior study that surveyed software ecosystem developers shows that two-thirds of the provider developers are interested in knowing whether their client packages adopt the latest release (Haenni et al., 2014). Therefore, it is important to understand the frequency and the speed with which client packages adopt same-day releases.

Approach. In this RQ, we study three measures regarding the adoption of same-day releases by client packages, namely the *implicit update rate*, the *explicit adoption frequency*, and the *adoption lag*. To establish a baseline, we also study the explicit adoption frequency and the adoption lag of prior-to-same-day and regular releases. In the following, we describe each of these measures:

Implicit update rate: To study the rate at which popular packages have their same-day releases implicitly adopted by their client packages, we calculate the proportion of client packages of a popular package that adopt a prior-to-same-day release using a version range statement that would accept patch updates of the provider (see Section 2.2 for a description of version range statements in npm). More formally, let $\text{CPSDR}_{p,i}$ be the set of client packages that adopt a prior-to-same-day release i of a popular package p. Now, let $\text{CPSDR}_{p,i}^{patch}$ be the subset of client packages in $\text{CPSDR}_{p,i}$ whose version range statement would implicitly accept patch updates. The implicit update rate of a same-day release $i+1$ of a popular package p is defined as:

$$IUR_{p,i+1} = \frac{|\text{CPSDR}_{p,i}^{patch}|}{|\text{CPSDR}_{p,i}|}$$

The implicit adoption rate provides a measure of the proportion of client packages that will automatically adopt a same-day release of a popular provider package without having to change the current versioning statement. By analyzing the implicit adoption rate, popular provider packages can reason about the proportion of their client packages that will immediately adopt their same-day releases.

Explicit adoption frequency: To understand the frequency at which same-day releases are explicitly adopted by a client package, we calculate the proportion of same-day, prior-to-same-day, and regular releases of the popular provider packages that are explicitly adopted by a client package. In our adoption frequency analysis, we focus on explicit adoption only. Explicit adoptions require the versioning statement to be changed by the client package (see Section 2.2). An explicit adoption represents a deliberate decision of the client to update the provider to a newer version. We use Fisher's Exact Test with Holm-Bonferroni correction to determine whether the difference in the proportion of same-day releases that are explicitly adopted is independent of the proportion of prior-to-same-day and regular releases.

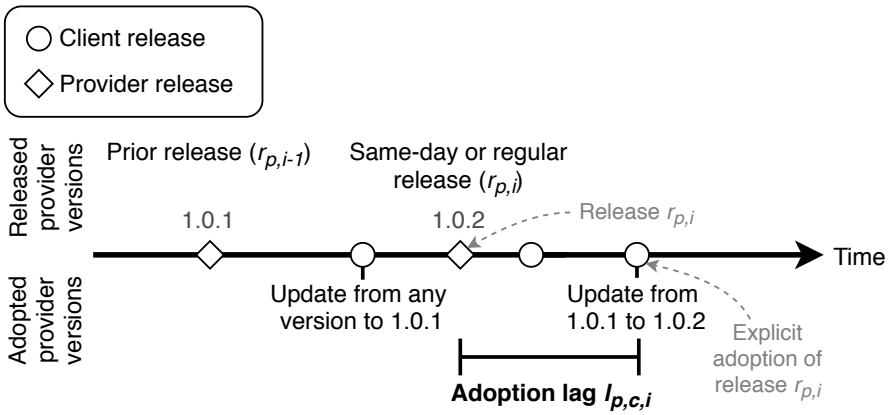

Fig. 14: Adoption lag of a same-day release.

In addition, we study the proportion of adoptions of prior-to-same-day releases that occur when the same-day release is already published. Our objective with this latter analysis is to determine whether provider packages need to improve the communication of same-day releases to client packages.

Adoption Lag: We study the adoption lag of same-day and regular releases. The adoption lag $l_{p,c,i}$ represents the period spanned between the release $r_{p,i}$ of the provider package p and the explicit update from the release $r_{p,i-1}$ to the release $r_{p,i}$ by a client package c, where $r_{p,i-1}$ is the provider release that precedes $r_{p,i}$ (see Figure 14). In essence, the adoption lag measures the length of the period during which a client package remains using the previous provider's release after the next provider release is published. If the introduced changes in same-day releases are important to client packages, we should observe a fast adoption of such releases. The reason for accounting for explicit adoption only is that implicit adoptions occur automatically right after the provider release is published. Also, explicit adoption is a well-thought action from the client package, which suggests that client packages have a clear interest in the explicitly adopted provider release. For example, supposing a client package c that explicitly adopts version 1.2.2 of a provider package p, the following sequence of actions needs to be performed for an explicit update:

1. Provider p publishes a release in npm (e.g., version p@1.2.3)
2. Client package c updates the versioning statement (e.g. from "p": "1.2.2" to "p": "1.2.3")
3. Client package c publishes a new release in npm.

We measure the adoption lag in terms of the number of hours elapsed (time lag) and the number of client releases (release lag). For example, we can either measure the elapsed time between the release $r_{p,i}$ and the adoption of $r_{p,i}$ or, alternatively, we can measure the number of client releases that are

published in the meantime between the release $r_{p,i}$ and the adoption of $r_{p,i}$. In addition, the release lag is measured in terms of major, minor, and patch releases of the client package (see Section 2.2). In our calculation of major, minor, and patch releases, we account only for the higher-level releases (major, minor, and patch). For example, if the client package publishes the release 1.2.3, 1.2.4 (patch), and 1.3.0 (minor) before the explicit provider update, we only account for the higher-level minor release (1.2.4 to 1.3.0). To evaluate the difference between the adoption lag of same-day and regular releases, we use the Wilcoxon Signed-rank test ($\alpha = 0.05$). We assessed the magnitude of the difference with the Cliff's Delta estimator of effect size. Finally, we clarify that adoption lag is different from technical lag (Zerouali et al., 2019) and dependency freshness (Cox et al., 2015) (measures that are commonly used in software ecosystem studies). While the adoption lag measures the lag between a provider package's release and the adoption of this release by a client package, both metrics as proposed by Zerouali et al. (2019) and Cox et al. (2015) measure the lag between the latest provider release and the currently adopted version of this provider by a client package.

Result 11: ***In general, clients of a popular provider package will implicitly adopt a same-day release of this provider.*** Table 7 shows the distribution of the implicit update rate: the median proportion of client packages that implicitly adopt a same-day release of a provider package is 100%. Indeed, packages in npm frequently use version ranges that accept patch updates to specify their dependencies (Decan and Mens, 2019). Also, the explicit adoption frequency of same-day releases is 18% lower than that of regular releases. While 39% of the same-day releases are explicitly adopted by a client package, 30% of the prior-to-same-day and 57% of the regular releases are explicitly adopted. The differences are both statistically significant (p-value $= 2.2 \times 10^{-16}$ for Fisher's Test, Holm-Bonferroni corrected). Moreover, we observe that client packages might be inadvertently adopting the prior-to-same-day when, in fact, the same-day release is already published. In total, 14% of the explicit adoptions of a prior-to-same-day release occurs when the same-day release is already published. As the difference in the number of same-day, prior-to-same-day, and regular releases that have a release note is at most 5%, we conjecture whether popular packages use other means to signal to client packages that a same-day release is available. In particular, we investigate whether prior-to-same-day releases are annotated with a deprecation message. We observed that only 3% of the prior-to-same-day releases that are adopted when a same-day release is already published are flagged as deprecated. In fact, a closer look reveals that only 7% of all prior-to-same-day releases are ever flagged as deprecated. We thus conclude that development teams of popular packages should make an effort to signal to client packages that there is a same-day release that addresses issues in the prior-to-same-day release (e.g., by improving release notes and deprecating the prior-to-same-day releases).

Table 7: The distribution of the implicit adoption rate.

	Min.	Q1	Median	Mean	Q3	Max.
Implicit adoption rate	0.0%	85.2%	100.0%	84.2%	100.0%	100.0%

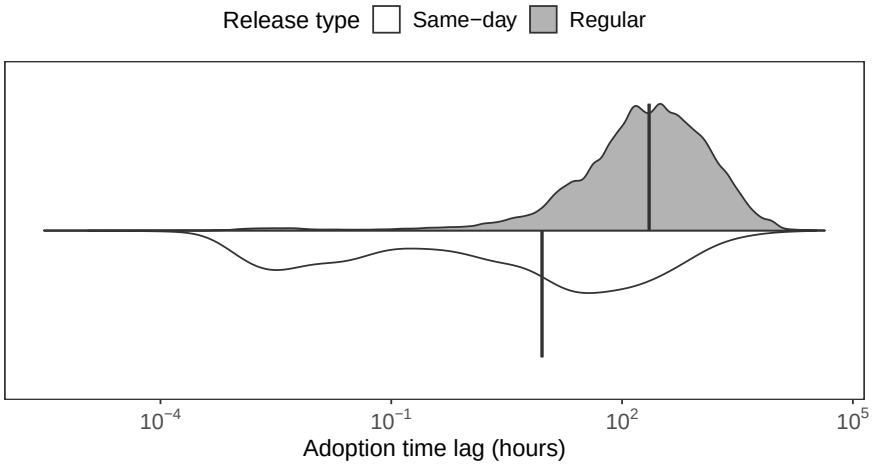

Fig. 15: Adoption lag for same-day releases.

Result 12: ***Same-day releases are explicitly adopted faster than regular releases.*** The median adoption time lag for same-day releases (9 hours) is approximately 25 times smaller than the median for regular releases (224 hours). Figure 15 shows the distribution of the time lag for same-day and regular releases and Table 8 summarizes its values. The difference in the distribution between same-day and regular releases is statistically significant (p-value $= 2.2 \times 10^{-16}$, Holm-Bonferroni corrected) and the effect size is large ($|d| = 0.626$). We also observed that 61% of the explicit adoption of same-day releases occurs in less than 24 hours, which shows that many explicit adoptions of same-day releases are performed relatively fast in comparison to regular releases (the overall median explicit adoption time lag is 1,102 hours). We conjecture that such fast adoptions are driven by the bug fixes that are incorporated in same-day releases (see Section 4).

In terms of release lag to adopt same-day or regular releases, the difference between same-day and regular releases is not statistically significant for patch releases (p-value < 0.05, Holm-Bonferroni corrected). For minor and major releases, the differences are statistically significant (p-value < 0.05, Holm-Bonferroni corrected) and the effect sizes are negligible ($|d| = 0.055$) and small ($|d| = 0.192$), respectively. The median adoption lag for both same-day and regular releases is one release (see Table 9).

Table 8: Summary of the adoption time lag for same-day and regular releases.

Release type	Min.	Q1	Median	Mean	Q3	Max.
Same-day	< 0.01	0.03	8.77	173.08	76.75	33,684.56
Regular	< 0.01	65.20	224.40	693.70	714.50	43,546.10

Table 9: Summary of the number of patch, minor, and major releases to explicitly adopt a same-day and a regular release.

Release level (client)	Release type (provider)	Min.	Q1	Median	Mean	Q3	Max.
Patch	Same-day	1.0	1.0	1.0	1.2	1.0	22.0
	Regular	1.0	1.0	1.0	1.4	1.0	453.0
Minor	Same-day	1.0	1.0	1.0	1.2	1.0	9.0
	Regular	1.0	1.0	1.0	1.1	1.0	80.0
Major	Same-day	1.0	1.0	1.0	1.1	1.0	8.0
	Regular	1.0	1.0	1.0	1.2	1.0	25.0

How do client packages adopt same-day releases?

We analyzed the frequency and the speed with which same-day releases are implicitly adopted by client packages. We observed that:

- The explicit adoption rate of same-day releases is 18% lower than that of regular releases.
- The median adoption time lag for same-day releases (9 hours) is significantly larger than the median time for regular releases (224 hours).

6 Discussion

In this section, we discuss the findings presented in Section 5. We first discuss the need for client packages to better evaluate the trade-off between the value and risk of same-day releases and how this task is currently being automated. Next, we discuss that popular package developers should plan for publishing same-day releases. We finally discuss the need for provider packages to improve the awareness of how their releases will impact client packages.

Implication 1: ***Third-party tools that support the automation of dependency management (e.g., Dependabot) should consider explicitly flagging same-day releases.*** Our preliminary study shows that same-day releases that are accompanied by release notes frequently contain important bug fixes. It is reasonable to assume that when developers fix a bug, document the fix, and publish a new release in less than 24 hours, they wish that this release will be absorbed by client packages as soon as possible. On the other hand, such a quickly implemented bug fix might have been complex,

involving several code changes, thus rendering the new release particularly error prone (Nagappan and Ball, 2005). Indeed, we observed that 39% of the same-day releases have a higher code churn compared to their associated prior-to-same-day release. We also observed that 32% of the same-day releases are even followed by another same-day release. Hence, the important trade-off between the value of a same-day release (e.g., the bug fix) and its potential risks (e.g., the chances of having new bugs) should be carefully evaluated by client packages. However, in order to do so, client packages need to be able to distinguish same-day releases from regular patch releases. Given the size of the npm ecosystem and the frequent use of version range statements, it is difficult for client packages to keep track of provider updates and the introduced changes in these updates. This challenge has led to the development of tools such as Dependabot,[19], which automates dependency management. Dependabot, which has been recently acquired by GitHub, inspects the list of dependencies of a project, searches for outdated dependencies, and automatically opens individual pull requests to update each dependency (Figure 16).

Developers of tools such as Dependabot should consider explicitly signaling same-day releases to client packages (especially if the code churn is large). This signalling would highlight the need for client packages to carefully evaluate the trade-off between release value and risk. If client packages become aware that a certain release is a same-day release, then they can make more informed decisions. For instance, client packages might decide to adopt the release preceding the prior-to-same-day release, they might decide to adopt the same-day (e.g., if they trust the provider package or if they desperately need a hotfix), or they might even decide to wait a couple of days to ensure that no other same-day will follow the current one. Dependabot, in particular, shows the release notes associated with a dependency update. However, there is no clear indication of whether a release is same-day. In our preliminary study, we often needed to read further documentation (e.g., issue reports) in order to fully understand the changes introduced in the same-day release. Therefore, we encourage developers of same-day releases to strive for more cohesive and clear release notes. If time pressure is too strong, developers of same-day releases can include a pointer to an external URL (e.g., GitHub or the package's website) where release notes can be carefully written after the release. The better the release notes, the easier it is for client package developers to decide whether to upgrade to a same-day release or not. Finally, if a same-day release fixes a critical bug, then we recommend developers to deprecate the prior-to-same-day release in order to encourage users to perform the update. In some software ecosystems, such as Rust, a package's release can be "yanked" – a concept that is similar to deprecation. Nonetheless, while a deprecated release can still be adopted by a client package, the package manager prevents the resolution of the provider package to a yanked version. Hence, as npm does not provide a similar feature, automated tools for dependency management

[19] `https://dependabot.com`

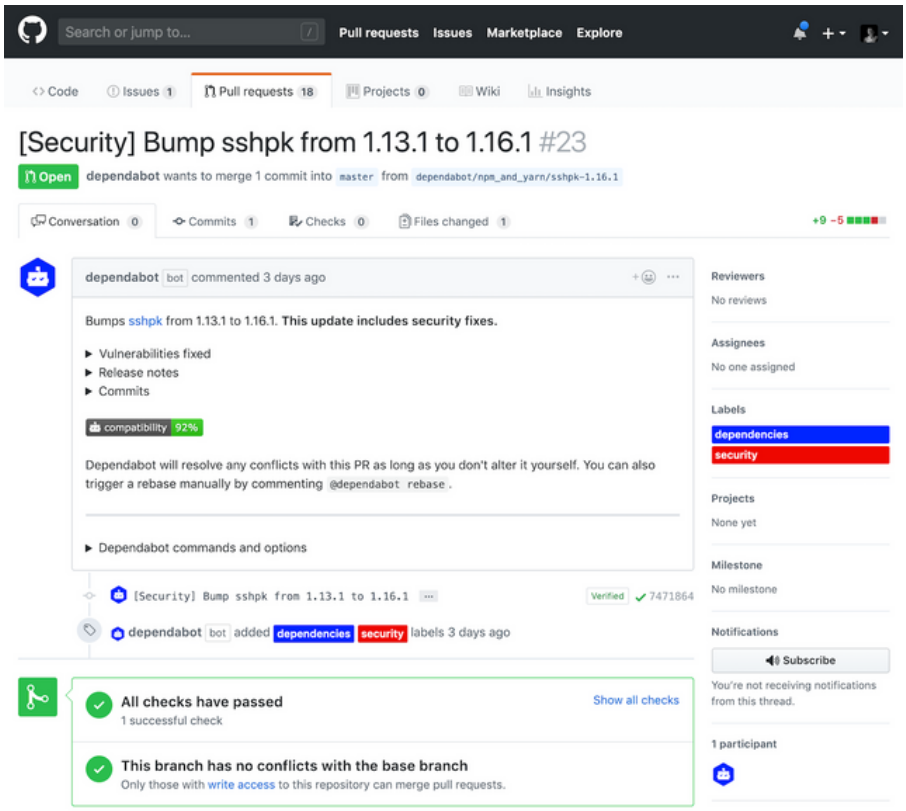

Fig. 16: An example of a pull request created by Dependabot. The pull request includes a list of the vulnerabilities fixed, the release notes, and commits associated with the dependency update. Image extracted from `https://dependabot.com`.

should prevent a client package to update towards a provider's release that contains a critical bug (as stated in the provider's release note).

Implication 2: ***Popular packages should strive for optimized release pipelines.*** In RQ1, we observed that the vast majority (96%) of the popular npm packages publish at least one same-day release and that 26% of all releases of the popular packages are same-day. We also observed a strong correlation between the number of releases and the number of same-day releases of a package. In other words, as popular packages evolve and publish new releases, it is likely that some of those new releases will be same-day releases. In addition, we observe a median interruption rate of 22% per popular package. That is, for every 5 regularly scheduled releases of a popular package, one is interrupted by a same-day release or a sequence of same-day releases. Therefore, popular npm packages will benefit from being able to quickly roll out a same-day release.

Although the adoption of release automation tools is a well-established software engineering practice, traditional automation tools like TravisCI[20] do not tackle ecosystem-specific problems. For instance, the constant updating of provider packages and configuration files by client packages can prevent build mechanisms to take full advantage of caching, since new provider versions or new configurations need to be reloaded. In response to this problem, npm has developed and improved upon the `npm ci` functionality[21], which aims to speed up build processes by skipping several user-oriented features that are part of the `npm install` functionality. Moreover, the `npm ci` functionality avoids updates to configuration files and individual dependencies during builds, as dependencies are deemed frozen and read from either the `package-lock.json` or `npm-shrinkwrap.json` file. Therefore, we encourage popular npm packages to have, in addition to their regular release pipeline, an optimized release pipeline for same-day releases. This optimized release pipeline should be configured to: 1) take the code of a release candidate as input, 2) be able to run only essential tests and build steps over the performed changes (e.g., with the help of `npm ci`), 3) deprecate the prior-to-same-day release, 4) generate release notes that clearly list the bug fixes and that explicitly flag the release as a same-day one (e.g., "hotfix"), and 5) promptly deploy the same-day release into production. Also, in RQ2, we observe that the update of provider packages is performed in only 17% of the same-day release. Therefore, it seems feasible to optimize this release pipeline by reusing previously installed versions of provider packages (i.e., by avoiding new installations of a provider package). As reported by prior research, having a backup release pipeline that supports expediting a new release is a practice implemented by popular software systems both in the commercial (e.g., Windows (Czerwonka et al., 2011)) and the open-source (e.g., Mozilla (Adams et al., 2015)) realms. Finally, we highlight that same-day releases invoke the notion of release readiness (Al Alam et al., 2014), which is an active research topic but with no commonly agreed upon guidelines as to how developers should reason about readiness. We thus call for future research to define proper ways to determine release readiness in software ecosystems.

Implication 3: ***Future research should design scalable, ecosystem-ready tools that support provider packages in assessing the impact of their code changes on client packages.*** In RQ3, we observed that, although the adoption rate of same-day releases are smaller than the adoption rate of regular releases, same-day releases are adopted significantly faster. In fact, 61% of the explicit adoption of a same-day release occurs in less than 24 hours after the same-day release was published. Since same-day releases are quickly adopted by client packages (either implicitly by the usage of version range statements or even by explicit updates), it is important for popular provider packages to understand how certain code changes will impact client packages. By combining the assessment of the importance of a same-day release

[20] `https://travis-ci.org`

[21] `https://blog.npmjs.org/post/171556855892/introducing-npm-ci-for-faster-more-reliable`

to client packages and how client packages are impacted by the current changes, popular provider packages can make a more informed decision about when exactly to release (or even whether field testing should be emphasized for that release).

Currently, the support for assessing the impact of a provider release on its client packages is limited and the existing solutions (e.g., regularly crawling the npm registry) are not sufficient for a precise and timely assessment. Prior research proposes to use the test suite of client packages to assess the impact of the changes introduced in a new provider package version (Mujahid et al., 2020; Chen et al., 2020; Mezzetti et al., 2018). However, such tools are designed to attend to the needs of client packages and would thus need to be extended or adapted to fulfil the needs of provider packages. We foresee that scalability would be a major challenge that provider packages would face when trying to use clients' test suites to assess the impact of their own changes. The proposed tools by prior research require that a number of client packages run their test suites against the changes introduced by the provider until it is possible to determine with a certain level of confidence whether the provider introduced any problematic change. For popular packages that are about to publish a same-day, this approach might not work, since running a high number of client tests against the introduced changes can take a large amount of time. Moreover, the proposed tools would require provider packages to know what their client packages are, which is difficult in an ever changing environment as npm. To overcome these challenges, provider packages would need to select a representative set of test cases from client packages and run their own changes against this representative set. Also, provider packages would need to periodically crawl the npm registry to identify their client packages. Developers of provider packages would benefit from research endeavours on the design of scalable, ecosystem-ready tools that can inform whether (and how) code changes will propagate to client packages. In other words, change impact analysis(Arnold, 1996) at the ecosystem level.

7 Threats to validity

Internal validity. *Controlling factors related to a package's evolution:* In RQ2 and RQ3, we investigate different metrics with respect to the code changes and the adoption of same-day releases, respectively. To establish baselines, we compare the same set of metrics between same-day and prior-to-same-day releases, as well as between same-day and regular releases. Beyond serving as a baseline, the comparison between same-day releases and prior-to-same-day releases also serves the objective of controlling the influence of factors related to the package's evolution on our results (e.g., the maturity level of a package or the number of developers contributing in a certain period), since these releases are both published within a period of 24 hours. As observed in our results of RQ2, usually the effect of comparisons performed between same-day and regular releases is larger than the effect of comparisons performed

between same-day and prior-to-same-day releases, which can be explained by the influence of such evolution-related factors.

The number of updated providers in same-day releases: In RQ2, we measure the number of provider updates that occur in a same-day release. However, same-day releases are published quickly after the prior release and, therefore, it is unlikely that a provider will publish a new release in the meantime between the prior-to-same-day release and the same-day release. This fact explains the large effect sizes found in the comparison between the number of provider updates in same-day and prior-to-same-day, as well as between same-day and regular releases. Nonetheless, we do not focus on the comparison of the number of updated providers between release types but, rather, we focus on understanding the rationale behind the update of providers in same-day releases.

Construct validity. *Identification of release notes:* In our preliminary study, we identify release notes by matching file names with a regular expression. Our manual analysis allowed us to validate the correctness of this method to identify changelog files (i.e., files that contain release notes). In particular, we observed that all sampled files were correctly identified as a changelog file and are associated with the correct release. Although correct, we cannot ensure that our method has perfect recall. For instance, it is possible that certain projects use an unconventional naming scheme for their changelog files, which will not be captured by our regular expression. In addition, certain projects might decide to store release notes exclusively on external locations (e.g., a GitHub repository or project website with documentation).

We also verified whether release notes are stored in files whose name starts with "readme" or "README" strings. We manually analyzed the difference (per pair of adjacent releases) of the contents of a random sample (±5% C.I. and 95% C.L.) of such readme files to determine whether any release notes were added. Although we observed that a release note was added to 9.2% of the analyzed readme files, those files typically contain information that is related to the package's documentation (e.g., package description, installation and usage instructions), deprecation notices, and information on build status and test coverage. We observed that the main keywords that are used to indicate that the content that is added to the readme is a release note are "changelog" and "release history". To validate whether searching for such keywords is a suitable approach to identify release notes in a readme file, we manually examined 100 readme files that contain either the term "changelog" or "release history". We observed that the accuracy of the keyword search-based approach is 20%, with most of the false-negatives being explained by the fact that developers maintain references to "changelog" or "release history" files in their readme files (i.e., the readme file contains sentences such as "*The release history of this project is kept in the changelog.md file*"). Therefore, we are concerned that, by considering such files as changelog files, we would introduce more noise than signal. For this reason, we do not consider the release notes that can occasionally be found in those files.

Operational definition of same-days: In our data selection process, we select packages that have less than 50% of all releases as same-day releases. By applying such a selection criterion, our objective is to exclude from our analyses those same-day releases that are part of a regular release schedule. Same-day releases that are part of a regular release schedule resemble regular releases and, therefore, may not reflect the effects of a time restriction that we are interested in studying. Nonetheless, a different proportion other than 50% can be chosen as a threshold value to this selection criterion, depending on how conservative the definition of regular release schedule is. In specific, by assuming a 45% and a 55% threshold value as a selection criteria, we observed a difference of 100 (3%) and 216 (7%) selected packages compared with the number of selected packages using the 50% threshold value.

External Validity. Because we collected data exclusively from npm, our findings may or may not generalize to other ecosystems. Although npm is representative in size, each software ecosystem has its intrinsic characteristics. We note that the goal of this study is not to build theories around same-day releases that would apply to all software ecosystems. Rather, our study is an important first step towards a deeper understanding of how packages in the npm ecosystem publish same-day releases. Therefore, we acknowledge that additional studies are required in order to further generalize our results.

8 Conclusion

Same-day releases are important to maintain the ecosystem in a functional state. Although prior studies investigated the occurrence of same-day releases (Kerzazi and Adams, 2016; Lin et al., 2017; Hassan et al., 2017), this is the first study about same-day releases in a software ecosystem platform. We used data from the npm ecosystem to understand how often same-day releases are published, the changes that are performed in such releases, and how client packages react to the same-day releases of their providers. Based on our empirical observations, we provide implications that can help development teams to reason about the time constraint of same-day releases and to improve the management of development activities in such releases.

We observed that, despite the restricted time frame within which same-day releases are developed, these releases introduce important changes. Also, 96% of the popular packages published at least one same-day release. A non-negligible proportion of the same-day releases modify more lines of code compared with prior-to-same-day and regular releases. Finally, many same-day releases are usually promptly adopted by client packages (i.e., in a few hours or in the next client release).

Our findings yield three main implications. First, large changes implemented in a quick manner can render some same-day releases particularly error prone. Indeed, same-day releases are often patched: 39% of the same-day releases are followed by another same-day release. Hence, client packages need proper support to assess the trade-off between the value and the risk

of adopting a same-day release. Second, the high rate of same-day releases in npm suggests that it is unavoidable to popular provider packages to eventually publish a same-day releases. Hence, popular packages will benefit from having a release pipeline that is able to process same-day releases, which might require specific configurations and tooling. Finally, since same-day releases are generally quickly adopted by client packages, provider packages will benefit to timely understand how the implemented changes impact their client packages. Our empirical observations contribute to the advance of the research concerning release management in software ecosystems. In particular, our study complements prior studies that relate same-day releases with releases that address urgent issues in the prior release.

A Changes introduced in same-day releases

In Table 10 of this appendix, we provide a detailed description of each of the identified notable changes in the same-day release notes. See Section 4 for a detailed description of our manual analysis of same-day release notes.

Table 10: Detailed description and example quotes of the notable changes in same-day releases.

Category	Description	Example quotes
Change business logic (18%)	The package's behaviour is modified so that the correct business rules are followed.	"Start webserver and browsers after preprocessing completed" (release note of karma@0.12.35) "changing attributes on '<option>' elements will now have the correct consequences" (release note of jsdom@1.3.1) "Fix: 'no-return-wrap' rule missing from index.js" (release note of eslint-plugin-promise@3.2.1) "'inspect()' checksum of empty file is now 'null'." (release note of fs-jetpack@0.5.2)
Correctly integrate provider (8%)	A provider package is correctly integrated into the client package.	"properly add pnp plugins" (release note of poi@12.4.2) "When ipfs-api runs inside Electron (...) the request logging is breaks [sic] the lib. It will be better to check whether res.req is set instead of relying on detect-node" (pull-request of ipfs-api@22.2.1) "Corrected CSSO API usage" (commit #df6f48 of package svgo@0.6.6)
Avoid crashing (7%)	A change is performed to avoid unrecoverable failures.	"Fixed: 'declaration-block-no-ignored-properties' now longer crashes on nested rules." (release note of stylelint@6.3.1) "Fix bug where using 'Object.create(null)' as a rejection reason would crash bluebird" (release note of bluebird@2.5.2)
Refactoring (7%)	A refactoring operation is performed.	"Tidy up build process" (release note of buble@0.4.23) "rename 'startBytes' to 'inBytes' and 'endBytes' to 'outBytes"' (release note of svgo@0.1.2)
Add new feature (6%)	A new feature is made available in the release.	"add a clean command to wipe out local dirs" (release note of gatsby@2.1.8) "src: add SET_SIZEOF and SET_ALIGNOF macros" (release note of ref@0.3.5)

Category	Description	Example quotes
Change configuration option (5%)	The set of values for a configuration option is modified.	"add altNames + csrConfigFile options to createCertificate" (release note of pem@1.14.1) "Fixed validation of 'webpack.styles' config to allow it to be set to 'false'" (release note of nwb@0.16.2) "Add "–host" flag to 'firebase experimental:functions:shell' (...)" (release note of gatsby@2.2.1)
Fix UI error (5%)	A change is performed to fix a failure that causes a User Interface (IU) malfunctioning.	"The previous Chrome deprecation fixes broke spatial positioning in Safari" (release note of howler@2.0.12) "UI now responds to touch events, and works on mobile devicesUI now responds to touch events, and works on mobile devices" (release note of react-select@0.2.12)
Revert previous release (5%)	The release is published in an explicit attempt to revert a change introduced in the previous release	"Incorrectly published the previous version from the wrong branch. That version doesn't contain all the work that is [sic] was supposed to. You shold [sic] use this version instead." (release note of bookshelf@0.14.1) "please ignore 0.20.0 (...)" (release note of rework@0.20.1) "Fix package publish" (release note of vue-awesome@3.0.2)
Improve build (4%)	A change to improve the build process is introduced	"Remove source maps from build" (release note of zoroaster@3.6.1) "Tidy up build process" (release note of buble@0.4.23) "Reduce size of distributed build" (release note of radium@0.12.2)
Improve documentation (4%)	A change to improve the package's documentation is introduced	"README.md link to "popular projects" presets doesn't work" (closed issue #285 of package standard-changelog@1.0.17) "Add note about API permissions (...) Without the correct permissions you'll get an error (...) Ideally the plugin would check for permissions and raise a more helpful error, but a note in the readme should help until then" (closed issue report #13037 of package gatsby@2.3.6)
Version control chore (4%)	A VCS operation (such as a branch merging) is performed.	"Merged _diffDeep fix" (release note of package config@0.4.20) "fix(release): cherry-pick the release commit to master on success" (commit #6830e of package sweetalert2@7.26.19) "Merged a branch that needed to be merged" (release note of package saasdoc@1.3.1)
Fix bug caused by provider (3%)	A bug introduced by a provider package is fixed.	"update bili [one of the package's dependencies] to fix a transpilation bug" (commit #5e4c8 of package rollup-plugin-postcss@1.2.1) "It seems a rollup-pluginutils [one of the package's dependencies] failure" (issue report #303 of rollup-plugin-commonjs@9.3.2)
Fix typo (3%)	An incidental typing error is fixed.	"Skip comments between imports" (pull request #164 of package postcss-import@8.0.2) "Fix version number in CHANGELOG.md" (release note of package ecstatic@2.2.1)

Category	Description	Example quotes
Optimize performance (3%)	A performance optimization change is introduced.	"improve json parser speed" (release notes of body-parser@1.1.2) "(...) overhaul to LiveReload backend to make it faster and more robust" (release note of package budo@10.0.3)
Update provider package (3%)	A provider package is updated to a newer version.	"Update libphonenumber@8.10.1" (release note of package google-libphonenumber@3.2.1) "Update Tippex to ∧2.1.1" (release note of package rollup-plugin-typescript@0.7.3)
Address failed bug fix (2.5%)	A failed attempt to fix a known bug drives a new attempt, which results in a release.	"The bug it still there after updating to 1.1.9, please take care to do tests before closing issues (...)" (issue #1093 of package ng2-bootstrap@1.1.14) "Same memory leak fixes as 0.4.1, properly applied to batch() operations too" (release note of package leveldown@0.4.2)
Deprecated package's feature (2.5%)	A package's feature is deprecated.	"inform user that 'start-selenium' is deprecated" (release notes of selenium-standalone@3.0.3) "Deprecated - When using '<include>' with body content, nested body content is now passed in as 'String' property named 'body' (...)" (release note of package marko@2.0.4)
Add missing file (2%)	The release add an important file that was missing on the previous release.	"include missing file in publishing" (release note of conventional-recommended-bump@2.0.1) "Fix bug where deploying functions resulted in the error message "npm ERR! missing script: build" (...)" (release note of package firebase-tools@3.17.1)
Add newer runtime support (1.5%)	A change for supporting a new runtime environment (Node) is performed	"Support Node.js 0.10.16" (release note of package ioredis@1.3.6)
Add provider package (1.5%)	A new provider package is introduced	"add 'ignore' dep" (release note of package now@0.24.1) "Use rollup-babelUse rollup-babel" (release note of package sorcery@0.6.5)
Fix dependency failure (1.5%)	A change to cope with a failure in a dependency loading is introduced.	"When installing the last version via NPM (...) signal-exit module seems to be missing." (pull-request #3186 of package stylelint@9.1.1) "Fix AMD-related failures first appearing in v3.5.1" (release note of package mocha@3.5.2)
Fix lint error (1.5%)	A change to adjust static analysis error is introduced.	"fix linting errors" (release note of package svg-sprite-loader@3.7.3)
Add backward compatibility (1%)	A change to add backward compatibility support for a feature is introduced.	"Support prefixing for old flexbox implementations" (release note of package radium@0.12.2)
Add hardware support (1%)	A change to add a new hardware support is introduced.	"There were a few different issues with these "helpers": 'data.key' didn't account for international keyboards (...)" (release note of package slate-react@0.5.4)
Address security vulnerability (1%)	A change to address a security vulnerability is introduced.	"Fixed more security vulnerabilities" (release note of package mathjs@3.11.1)

Category	Description	Example quotes
Fix memory leak (1%)	A change to address a memory leak defect is introduced.	"Fix memory leak caused when passing String objects in as keys and values (...)" (release note of package leveldown@0.4.2)
Prevent error (1%)	A change is introduced to prevent some defect.	"Prevent error when destructured [sic] path is not in known globals" (release note of package eslint-plugin-ember@5.0.1)

References

Adams B, McIntosh S (2016) Modern Release Engineering in a Nutshell – Why Researchers Should Care. In: Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, IEEE, SANER'16, pp 78–90

Adams B, Bellomo S, Bird C, Marshall-Keim T, Khomh F, Moir K (2015) The practice and future of release engineering: A roundtable with three release engineers. Software, IEEE 32:42–49, DOI 10.1109/MS.2015.52

Al Alam SMD, Shahnewaz SM, Pfahl D, Ruhe G (2014) Monitoring bottlenecks in achieving release readiness: A retrospective case study across ten oss projects. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM'14

Arnold RS (1996) Software Change Impact Analysis. IEEE Computer Society Press

Bauer DF (1972) Constructing confidence sets using rank statistics. Journal of the American Statistical Association 67(339):687–690

Castelluccio M, An L, Khomh F (2019) An empirical study of patch uplift in rapid release development pipelines. Empirical Software Engineering 24(5):3008–3044

Chen L, Hassan F, Wang X, Zhang L (2020) Taming behavioral backward incompatibilities via cross-project testing and analysis. In: Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering, ICSE'20, p 112–124

Cliff N (1996) Ordinal methods for behavioral data analysis. Psychology Press, New-York, USA

Cox J, Bouwers E, van Eekelen M, Visser J (2015) Measuring dependency freshness in software systems. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 2, pp 109–118

Czerwonka J, Das R, Nagappan N, Tarvo A, Teterev A (2011) Crane: Failure prediction, change analysis and test prioritization in practice – experiences from windows. In: 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, pp 357–366

Decan A, Mens T (2019) What do package dependencies tell us about semantic versioning? IEEE Transactions on Software Engineering pp 1–1

Decan A, Mens T, Claes M (2017) An empirical comparison of dependency issues in OSS packaging ecosystems. In: Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER'17, pp 2–12

Erenkrantz JR (2003) Release management within open source projects. In: Proc. 3rd. Workshop on Open Source Software Engineering

Eski S, Buzluca F (2011) An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, IEEE Computer Society, Washington, DC, USA, ICSTW '11, pp 566–571

Fox A (2002) Toward recovery-oriented computing. In: Bernstein PA, Ioannidis YE, Ramakrishnan R, Papadias D (eds) VLDB '02: Proceedings of the 28th International Conference on Very Large Databases, Morgan Kaufmann, San Francisco, pp 873 – 876

Graves TL, Karr AF, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Transactions on Software Engineering 26(7):653–661

Haenni N, Lungu M, Schwarz N, Nierstrasz O (2014) A quantitative analysis of developer information needs in software ecosystems. In: Proceedings of the 2014 European Conference on Software Architecture Workshops, ACM, New York, NY, USA, ECSAW '14, pp 12:1–12:6

Hamilton J (2007) On designing and deploying internet-scale services. In: Proceedings of the 21st Conference on Large Installation System Administration Conference, USENIX Association, Berkeley, CA, USA, LISA'07, pp 18:1–18:12

Hassan S, Shang W, Hassan AE (2017) An empirical study of emergency updates for top android mobile apps. Empirical Software Engineering 22(1):505–546

Jaspan C, Jorde M, Knight A, Sadowski C, Smith EK, Winter C, Murphy-Hill E (2018) Advantages and disadvantages of a monolithic repository: A case study at google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ACM, New York, NY, USA, ICSE-SEIP '18, pp 225–234

Kajko-Mattsson M, Yulong F (2005) Outlining a model of a release management process. Journal of Integrated Design and Process Science 9(4):13–25

Kerzazi N, Adams B (2016) Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, IEEE, SANER'16, pp 574–583

Kikas R, Gousios G, Dumas M, Pfahl D (2017) Structure and evolution of package dependency networks. In: IEEE International Working Conference on Mining Software Repositories, MSR'17, pp 102–112

Lahtela A, Jäntti M (2011) Challenges and problems in release management process: A case study. In: Proceedings of the 2nd IEEE International Conference on Software Engineering and Service Science, ICSESS'11, pp 10–13

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014) Upset: Visualization of intersecting sets. IEEE Transactions on Visualization and Computer Graphics 20(12):1983–1992

Lin D, Bezemer CP, Hassan AE (2017) Studying the urgent updates of popular games on the Steam platform. Empirical Software Engineering 22(4):2095–2126

Louis M Rea RAP (2014) Designing and Conducting Survey Research: A Comprehensive Guide, 4th edn. John Wiley Sons

Mäntylä MV, Adams B, Khomh F, Engström E, Petersen K (2015) On rapid releases and software testing: a case study and a semi-systematic literature review. Empirical Software Engineering 20(5):1384–1425

Mezzetti G, Møller A, Torp MT (2018) Type regression testing to detect breaking changes in node.js libraries. In: 32nd European Conference on Object-Oriented Programming (ECOOP 2018)

Moreno L, Bavota G, Di Penta M, Oliveto R, Marcus A, Canfora G (2014) Automatic generation of release notes. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14), pp 484—495

Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '08, pp 181–190

Mujahid S, Abdalkareem R, Shihab E, McIntosh S (2020) Using Others' Tests to Identify Breaking Updates, p 466–476. MSR'20

Nagappan N, Ball T (2005) Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '05, pp 284–292

Romano J, Kromrey J, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? In: Annual Meeting of the Florida Association of Institutional Research

Schober P, Boer C, A Schwarte L (2018) Correlation coefficients: Appropriate use and interpretation. Anesthesia Analgesia 126(5)

Stol KJ, Ralph P, Fitzgerald B (2016) Grounded theory in software engineering research: A critical review and guidelines. In: Proceedings of the 38th International Conference on Software Engineering (ICSE'16), pp 120–131

Van der Hoek A, Wolf AL (2003) Software release management for component-based software. Software - Practice and Experience 33(1):77–98

Wittern E, Suter P, Rajagopalan S (2016) A look at the dynamics of the JavaScript package ecosystem. In: Proceedings of the 13th International Workshop on Mining Software Repositories, MSR'16, pp 351–361

Zerouali A, Mens T, Gonzalez-Barahona J, Decan A, Constantinou E, Robles G (2019) A formal framework for measuring technical lag in component repositories and its application to npm. Journal of Software: Evolution and Process

View publication stats