Graduation Thesis

Title

Trend Analysis of Mobile App Abandonment from User Metrics

Supervisor
Assistant Professor Olivier Nourry

Author
Kazuya Matsushita

February 9th, 2026

Department of Information and Computer Sciences,
School of Engineering Science,
The University of Osaka

Reiwa 7th Graduation Thesis

Trend Analysis of Mobile App Abandonment from User Metrics

Kazuya Matsushita

Abstract

Distinguishing between maintained and abandoned mobile applications is important for improving user experience on app marketplaces. However, prior research has rarely examined development abandonment in mobile application contexts, although it has been studied in the OSS ecosystem. Developer-centric indicators such as the Truck Factor (TF)—which measures how many developers are critical to a project's sustainability—and Truck Factor Developer Detachment (TFDD)—situations in which all developers counted by the TF become detached from a project—have been proposed and studied in the OSS ecosystem. We analyze 706 Android applications released on the Google Play Store and examine how these TF-based indicators relate to user metrics such as ratings and review activity. The results show that 52.5% of the studied applications experience TFDD, and many of them show slightly higher quantitative user metrics. These metrics, such as the number of installs and average rating, are insufficient to detect development abandonment. In contrast, qualitative analysis of user reviews reveals that applications experiencing TFDD receive more reviews expressing dissatisfaction or reporting issues. These findings highlight significant differences between applications experiencing TFDD and maintained applications. Consequently, review content can be used to distinguish development abandonment, thereby supporting better user experience on app marketplaces.

1 Introduction

Since the advent of smartphones, mobile applications have become integral to daily life for billions of people worldwide. With millions of applications released through major app stores such as the Apple App Store [1] and Google Play Store [2] [3, 4], mobile development constitutes a significant portion of the software ecosystem. As in any ecosystem, some applications thrive while others are abandoned by their developers.

Recent trends suggest that the pace of mobile application releases is accelerating, coinciding with the emergence of AI-assisted and agentic coding practices [5]. While these practices lower barriers to entry by enabling small teams or individual developers to rapidly produce and release multiple applications, they can also contribute to overcrowding in app marketplaces, potentially resulting in a larger number of applications that lack sustained maintenance.

For end users, discontinued development can have observable results. Applications become gradually unusable due to unresolved bugs, deprecated backend services, or incompatibilities introduced by platform updates. Users may need to migrate to alternative applications, often incurring non-trivial costs such as data loss, the effort to learn new interfaces, or adaptation to different workflows. When choosing applications in app stores, the increase of short-lived or poorly maintained applications tends to make it more difficult to identify reliable and sustainable apps and reduce overall user experience.

Despite these challenges, little work has systematically investigated the indicators that lead to mobile application abandonment. Prior research has largely focused on identifying determinants of app success. As a result, the relationship between developer activity and user experience remains relatively underexplored. In software engineering research, the *Truck Factor* (TF) has been proposed and used as a metric to assess a project's dependence on its key developers[6, 7, 8, 9, 10]. TF is defined as the minimum number of developers whose detachment would cause a project to stall or substantially slow down [6]. Building on this concept, *Truck Factor Developer Detachment* (TFDD) captures events in which all developers counted in the TF detach from the project, signaling elevated risks to continued development [9].

Prior studies have extensively applied TF- and TFDD-based analyses to open-source software (OSS) projects. For example, Ricca *et al.* [7] and Avelino *et al.* [8, 9] showed that many projects rely on a very small number of core developers to sustain development activities. Subsequent work has further reported substantial core developer turnover,

indicating TFDD is a common phenomenon in OSS ecosystems [11, 12, 10]. These findings collectively suggest that TF and TFDD are effective lenses for characterizing developer-related sustainability risks. However, existing evidence is largely largely based on general OSS contexts, and their applicability to mobile applications—where continuous updates are tightly coupled with user experience—remains insufficiently explored.

This study aims to reveal characteristics associated with app abandonment and to establish a foundation for the development of indicators in future research. To this end, we first analyze TF and TFDD in mobile applications and then examine their relationship with user metrics. While this work does not yet establish practical operational metrics, it provides insights for developing dynamic indicators that combine developer activity and user metrics. Indicators derived from these insights could ultimately improve app store experiences, supporting better recommendations and helping users make more informed choices when selecting applications.

Building on this integrated perspective, we formulate the following research questions:

- RQ1: How prevalent is TFDD among mobile applications, and when does it typically occur?
- RQ2: What are the characteristics of applications facing TFDD?
- RQ3: How do user reviews relate to TFDD events in mobile applications?

By answering these questions, this study aims to deepen our understanding of app development sustainability and provide insights that support the maintenance of a healthy mobile application ecosystem. In Section 2, we provide background on the core concepts of this study, in Section 3, we breakdown our dataset creation and our overall methodology, in Section 4, we show the results of our analyses, in Section 5, we discuss the implications and main takeaways of our findings, and finally we conclude the paper in Section 7.

2 Background

2.1 Continuous Development and Sustainability of User Experience

In mobile applications, user experience is evaluated not only by short-term quality but also by the ability to provide stable value over time. ISO 9241[13] defines usability in terms of effectiveness, efficiency, and satisfaction, emphasizing that user experience should be considered in relation to users, tasks, and evolving contexts of use.

Achieving consistently satisfactory user experience requires more than an initial release of a functional application[13]. Applications must be continuously updated to address evolving user needs, fix bugs, maintain compatibility with changing platforms, and incorporate feedback. This continuous development, together with iterative interaction with users, forms a cycle that is essential for sustaining usability and user satisfaction over the long term.

For end users, discontinuation of development has observable results. Applications that are no longer maintained gradually become unusable due to unresolved bugs, deprecated backend services, or incompatibilities introduced by operating system updates. Users may then be forced to migrate to alternative applications, often incurring non-trivial costs such as data loss, effort to learn new interfaces, and adaptation to different workflows or interaction patterns. At the same time, the increase of short-lived or poorly maintained applications contributes to overcrowding in app marketplaces, reducing overall user experience and making it harder to identify reliable and sustainable applications.

Taken together, these considerations highlight that continued development is generally desirable. While abandoned applications are not inherently "bad," distinguishing them from actively maintained applications is necessary to ensure a usable and sustainable mobile app ecosystem.

2.2 User Metrics as Indicators of Mobile App Success

Prior studies on mobile applications have primarily focused on identifying indicators contributing to app success [14, 15, 16, 17]. In this context, a range of user-centered metrics is commonly used. These include publicly available indicators such as download counts, review volume, and rating scores, as well as engagement metrics such as daily or monthly active users (DAU/MAU) and retention rates. These metrics are widely used to observe user behavior and app evaluation.

While these metrics are valuable for assessing application success, they primarily capture

outcomes visible to end users and do not directly reveal internal development processes, team dynamics, or risks associated with core developer detachment. Consequently, even applications that appear successful from a user perspective may face internal risks, such as core developers becoming inactive or leaving, which can ultimately degrade the user experience . This gap between external user indicators and internal development risks motivates the joint examination of user metrics and developer-level measures to better understand the conditions under which mobile applications are abandoned.

2.3 Truck Factor: Concept and Analysis in OSS

As discussed in the preceding subsection, continuous development activity is essential to sustain user experience in mobile applications. In this context, the *Truck Factor* (TF) is a widely used metric in software engineering that quantifies a project's vulnerability to the loss of developers. It is defined by Williams *et al.* [6] as the minimum number of core developers whose detachment would cause the project to stall or substantially slow down; these developers are also referred to as TF Developers. Hereafter, we distinguish core developers, who are the project's main contributors in general, and TF Developers, who are identified strictly based on the TF metric.

Building on this, Avelino *et al.* [9] proposed *Truck Factor Developer Detachment* (TFDD), which refers to events in which all TF Developers become inactive, potentially threatening the continued development of the project.

Several prior studies have used the TF to investigate software development activities in OSS projects. Ricca *et al.* [7] calculated the TF for 20 OSS projects and found that most relied on very few developers to maintain development activities. Avelino *et al.* [8, 9] conducted multiple studies investigating core developers in OSS projects using the TF and observed the same development pattern, where most projects rely on one or two TF Developers to take on the majority of the workload. Ferreira *et al.* [11] and Calefato *et al.* [12] also investigated core developer development patterns and observed significant core developer turnover, with Calefato *et al.* reporting that 45% of core developers completely stop contributing to OSS projects for extended periods. Nourry *et al.* [10] analyzed over 36,000 projects and found that 89% experienced TFDD at least once, often in early stages, and only 27% subsequently attracted new TF developers.

Several tools have been proposed to calculate the TF of software projects [7, 8, 18, 19]. Among these, the approach proposed by Avelino *et al.* [8] has been widely used in prior studies.

This approach is based on the concept of the Degree of Authorship (DOA), which captures a developer's ownership of a source file by considering whether the developer originally created the file, the absolute amount of their subsequent modifications, and their relative contribution compared to other developers [20, 21]. After computing DOA at the file level, ownership is aggregated across the project, and the minimum set of developers whose combined ownership accounts for at least 50% of the project's code base is defined as the TF developers.

To facilitate reproducibility, Avelino *et al.* released an official tool implementing this TF calculation framework [22]. In studies using TFDD, developers are treated as inactive if they have not contributed for a period of one year. This threshold has been shown to be appropriate in TF-related analyses [9].

2.4 Developer and User Perspectives on Mobile Apps

Understanding mobile app abandonment requires considering both developer activity, particularly TFDD, and user-facing indicators such as downloads, ratings, and reviews. While user metrics capture observable user behavior and evaluation, they do not reveal internal risks related to developer disengagement. In contrast, TFDD reflects the loss of core developers and the resulting risk of reduced maintenance or stalled development, which can precede application abandonment. By integrating these perspectives, this study adopts a unified analysis of user metrics and developer activity to better characterize the conditions under which mobile applications are abandoned.

2.5 Research Questions Motivated by User and Developer Indicators

Building on the preceding discussion, this study aims to reveal characteristics associated with mobile app abandonment and to establish a foundation for indicator development in the future. To this end, we jointly examine user-facing indicators and developer activity. These findings provide a foundation for developing dynamic indicators that integrate developer activity with user feedback, enabling more precise monitoring of potential app abandonment, potentially supporting improved recommendation mechanisms and a better application selection experience in app stores. This perspective motivates the following research questions:

RQ1: How prevalent is TFDD among mobile applications, and when does it typically occur?

Motivation. Based on observations from OSS, it can be expected that core developers also play a crucial role in the sustainability of mobile applications. For mobile applications, which require frequent updates for compatibility, security, feature enhancements, and bug fixes, the detachment of core developers can have particularly severe consequences.

RQ2: What are the characteristics of applications facing TFDD?

Motivation. This research question investigates the characteristics of applications experiencing TFDD. While TFDD is defined based on developer activity, it may be related to user-centered metrics such as install counts, ratings, and review volume. By comparing these metrics between Stable and TFDD applications, the study aims to explore potential associations between developer activity and observable patterns of user engagement and evaluation.

RQ3: How do user reviews relate to TFDD events in mobile applications?

Motivation. While RQ2 examined quantitative user indicators such as install counts, ratings, and review volume, it remains unclear how the content of user reviews relates to TFDD events. This question focuses on qualitative analysis of review sentiment and topics, which can provide insights into app abandonment and potentially reveal early signals of TFDD that are not captured by quantitative metrics.

3 Methodology

In this section, we describe our methodology to collect and filter our dataset and conduct our analyses.

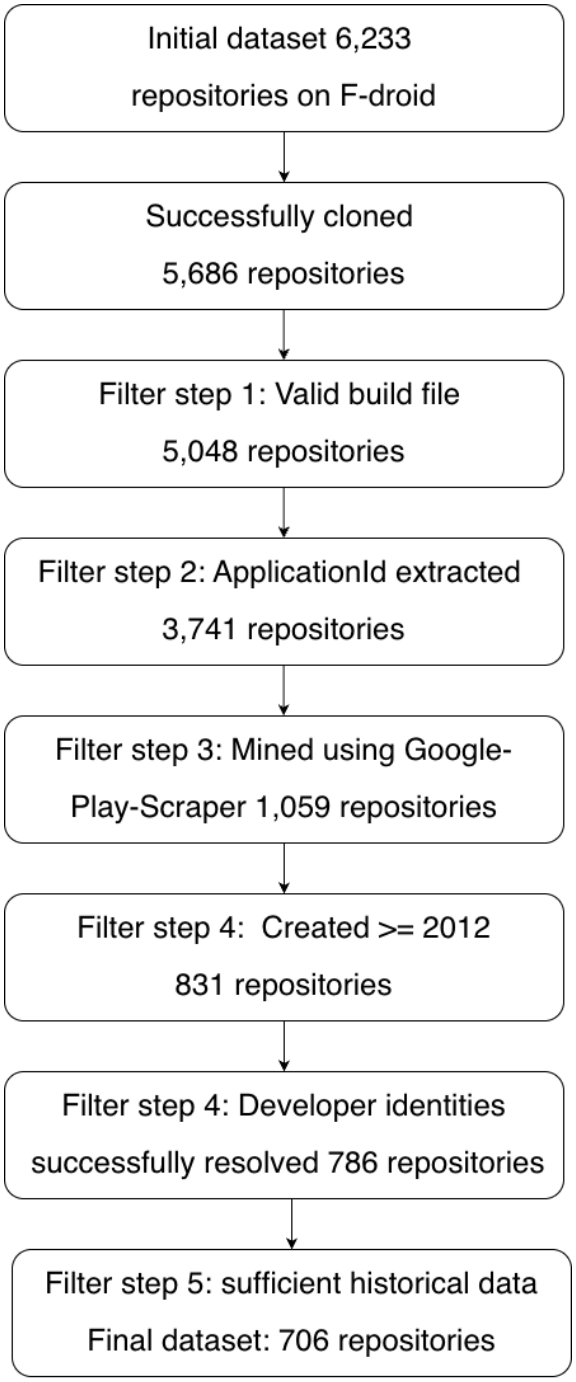


Figure 1: Overview of dataset filtering process and remaining repositories at each step.

3.1 Dataset Creation

To construct a dataset suitable for analyzing TFDD in mobile applications, we followed a multi-stage filtering process starting from a large collection of open-source Android projects (see Figure 1).

We first obtained a list of 6,233 open-source Android applications from the F-Droid ecosystem [23]. The corresponding source code repositories, primarily hosted on GitHub, were then programmatically cloned, resulting in 5,686 successfully retrieved projects. Repositories that could not be cloned due to missing, inaccessible, or relocated source code were excluded at this stage.

Next, to ensure that each repository corresponded to a valid Android application with an identifiable app entry in the Google Play Store, we examined the directory structure of each project to locate Gradle build configuration files. Projects lacking valid *build.gradle* files were excluded, as the absence of these files prevents extraction of the *applicationId*, which is required to uniquely link a source code repository to its corresponding marketplace entry. This filtering step reduced the dataset to 5,048 repositories.

From the remaining repositories, we extracted the *applicationId* from the Gradle configuration files. This identifier is required to uniquely link a source code repository to its corresponding application entry in the Google Play Store. Repositories for which the application identifier could not be extracted or resolved were removed, resulting in 3,741 candidate applications.

Using the extracted application identifiers, we queried the Google Play Store through the *google-play-scraper* library [24] to verify the existence of corresponding marketplace entries and simultaneously collect user metrics such as ratings, reviews, and install counts. Applications for which the scraper failed to retrieve marketplace information were excluded, as they could not be reliably linked to publicly available app store data. After this step, 1,059 repositories remained that could be reliably associated with marketplace data.

To ensure temporal relevance and sufficient historical context for longitudinal analysis, we applied two additional time-based filters. First, we excluded applications whose repositories were created before 2012, as the Android ecosystem and the Google Play Store were not yet fully established prior to this period [25]. This filtering step yielded 831 applications.

Next, during the preparation for TF analysis, we identified 41 repositories for which developer identities could not be consistently resolved due to ambiguous or missing author

information in the Git commit history. As reliable identification of developers is required for TF computation, these repositories were excluded from further analysis.

Finally, applications released in the most recent year (2025) were excluded due to the lack of sufficient development and usage history. After applying all filtering criteria, the final dataset used in our analyses consists of 706 Android application repositories.

3.2 TF and TFDD Calculations.

To identify applications that were abandoned by their developers (i.e., experienced a TFDD), we calculated the yearly TF for each application in our dataset. For each repository, we obtained its initial creation date and traversed the development history in one-year increments by checking out yearly snapshots using the *git checkout* command. At each yearly snapshot, we computed the TF using the tool proposed by Avelino *et al.* [22] and obtained the corresponding TF Developers.

As illustrated in Figure 2(a), the tool identifies TF Developers based on cumulative contributions, which may cause developers who were active in earlier periods but inactive in the target year to remain counted as TF Developers. To reflect actual developer activity in each year, we examined project histories using the *git log* command and verified whether each candidate TF Developer showed development activity during the corresponding year. Developers without recorded activity in that year were excluded from the set of active TF Developers.

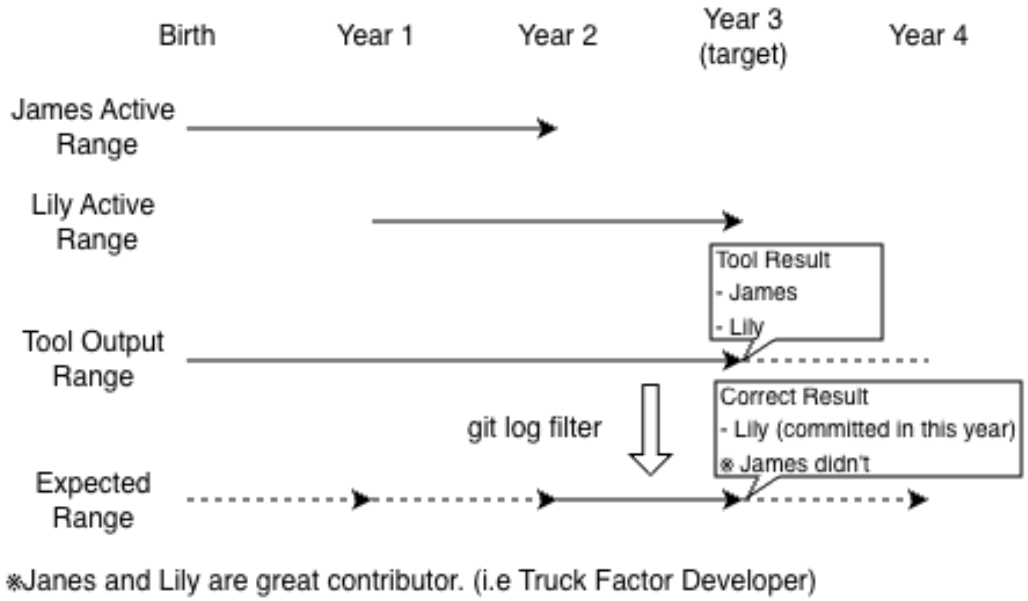
Figure 2(b) shows a concrete example of this process, where no developer exhibits activity in the target year after filtering. For any yearly snapshot in which no active TF Developers remained, we marked the application as having experienced a TFDD.

3.3 LLM-Based Classification of User Reviews

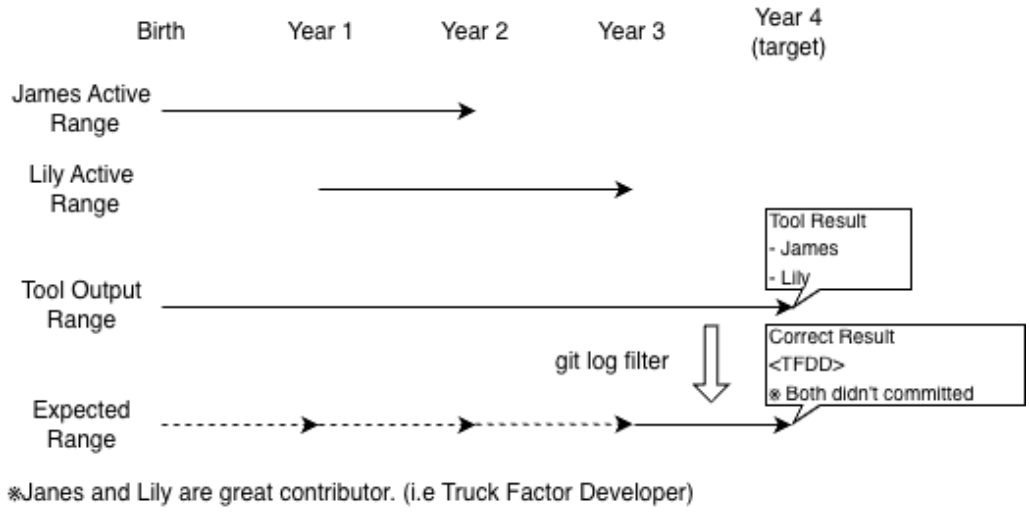
In RQ3, we investigate user reviews to identify patterns that may indicate whether an application is likely to remain stable or to experience a TFDD. Specifically, we conduct a qualitative analysis to classify the primary purpose expressed in each user review.

For this analysis, we employed a predefined set of three functional categories: `Feedback`, `Feature`, and `Bug`. These categories were defined as follows:

- **Feedback**: A user provides general opinions or impressions about the application (e.g., *"Great browser"*).



(a) Yearly TF correction example.



(b) TFDD example in the target year.

Figure 2: **Yearly TF correction and a TFDD example.**
(a) Only Lily shows development activity in the target year, and James is excluded after filtering.
(b) Neither James nor Lily shows development activity, resulting in no active TF Developers (TFDD).
Note that developers with contributions too small to be counted in the TF are already excluded in the tool output.

- **Feature**: A user requests a new feature or improvement (e.g., *"I wish it had a translate feature like Google does..."*).
- **Bug**: A user reports a malfunction or unexpected behavior (e.g., *"I can't import PDF files. Format Exception: Unexpected extension byte..."*).

When a review contained multiple intents, we applied a priority rule in which `Bug` and `Feature` labels took precedence over `Feedback`, reflecting their higher relevance to actionable development issues.

Since the classification of a large-scale review dataset relies on LLM-based labeling, careful prompt design and refinement were required prior to the full-scale analysis. To support prompt tuning, we randomly sampled 500 user reviews from the collected dataset.

These sampled reviews were initially labeled using ChatGPT-4 [26] to obtain a rough baseline annotation, without assuming the correctness of the generated labels. Two authors then manually inspected and corrected these annotations. The manually corrected labels were treated as reference annotations and used solely to iteratively refine and validate the LLM prompting strategy, while the predefined review categories remained unchanged.

Prompt refinement was repeated until the agreement between the LLM-generated labels and the manually corrected annotations exceeded 80%. Once this threshold was reached, the prompt configuration was fixed for the subsequent analysis.

For the analysis, we first collected all user reviews from applications that experienced at least one TFDD event, totaling 110,106 reviews, and randomly sampled an equal number of reviews from applications that never experienced a TFDD to enable a balanced comparison. Using the finalized prompt, refined through the previously described LLM prompting strategy, we then employed the GPT-4o mini model via OpenAI's API [27] to classify all reviews . The GPT-4o mini model was chosen for its favorable balance between classification quality and computational cost, enabling efficient large-scale annotation.

Following the functional classification, we reused the same dataset to perform a second qualitative analysis aimed at identifying the overall sentiment of each review, in order to investigate whether the emotional tone expressed by users differs between applications that experienced TFDD and those which did not, with the LLM assigning one of three sentiment labels—`Positive`, `Neutral`, or `Negative`—based on the dominant emotional tone; the sentiment analysis followed the same prompt refinement procedure as the functional analysis, using the same randomly sampled set of 500 reviews and manually corrected

annotations to validate the prompt prior to full-scale classification.

LLM Prompt Configuration. For the sake of reproducibility, we report the final prompts used for the LLM-based functional and sentiment classification of user reviews below.

Functional Classification Prompt

> You are an expert classifier for mobile application reviews published on the APP-store. Your task is to determine and label the intent or type of reviews provided to you. Here's a list of available review types/labels:
>
> - Feature : The feature label is used when a user requests improvements or specific functionalities for the application. - Bug : The bug label is used when a user encounters situation where the app does not behave as it should or as it is expected to behve. - Feedback : The feedback label is used when a user states their opinion about the app or their experience using the app.
>
> 1. Return only one word: Feature, Bug, or Feedback. 2. When a review contains multiple parts, the user's feedback has lesser priority than other labels. For example, given a sentence such as "The app is great but we think the UI is a bit lack luster", this sentence provides 1) feedback by saying that the app is great but also 2) feature by saying that the UI needs improvement. In this context, the feature label takes priority. Analyze the content provided to you and assign a label to each review contained in that csv file.
>
> Very short acknowledgment-style reviews such as "ok", "okay", or "thank you" should be labeled as Feedback.

Sentimental Classification Prompt

> You are an expert sentiment classifier for mobile application reviews published on the APP-store. Your task is to determine and label the emotional tone (sentiment) of user reviews.
>
> The available sentiment labels are:
>
> - Positive : The user expresses satisfaction, praise, or generally favorable emotions toward the app. - Neutral : The user provides a factual or mixed opinion without clear positive or negative emotion. - Negative : The user expresses dissatisfaction, frustration, or negative emotions toward the app.

Instructions: 1. Return only one word: Positive, Neutral, or Negative. 2. When a review contains multiple sentiments, choose the strongest overall sentiment that dominates the text. 3. Ignore sarcasm unless explicitly indicated. 4. Do not include explanations or any extra text — output must be exactly one of: Positive, Neutral, or Negative.

Analyze the review content provided in the CSV file and assign one of the above sentiment labels to each review.

4 Result

4.1 RQ1:How prevalent is TFDD among mobile applications, and when does it typically occur?

Approach. To better understand how vulnerable mobile applications are to abandonment, we first analyzed the annual trend in the number of TF developers across all applications between 2013 and 2024. Apps in which all TF developers were lost in a given year were marked as experiencing TFDD, and each app was then classified into one of two groups based on the occurrence of TFDD.

- **Stable:** Projects that have never experienced a TFDD.
- **TFDD:** Projects that experienced at least one TFDD event.

For the TFDD group, we further distinguished between two subcategories:

- **Survival:** Projects where TF Developers resumed their development activity or new TF Developers joined the project following a TFDD event.
- **Abandoned:** Projects where no TF Developer activity ever took place following a TFDD event.

After categorizing all projects, we further investigated the timing of TFDD events relative to the project's lifecycle, including the date of the initial commit, the release date on the app marketplace, and, for Abandoned applications, the date of the last update.
Results. Table 1 shows the proportion of projects that experienced a TFDD. Out of 706 applications, we found that 371 (52.5%) experienced at least one TFDD, while 335 (47.5%) never experienced any TFDD event.

Table 1: Stable group vs. TFDD group

Group	# of Apps	Percentage
TFDD	371	52.5%
Stable	335	47.5%
Total	706	100%

As shown in Table 2, among projects that experienced a TFDD, 158 (42.6%) were able to attract a TF Developer again and continue maintenance and development activities, while 213 (57.4%) were abandoned.

Table 2: Survival vs. Abandoned

Group	# of Apps	Percentage
Survival	158	42.6%
Abandoned	213	57.4%
Total (TFDD)	371	100%

Figure 3 shows the yearly distribution of mobile applications by the number of active TF developers, separated into Stable and TFDD groups. The horizontal axis represents the calendar year, and for each year applications are divided into the two predefined groups. The vertical axis indicates the number of active TF developers. For each year and group, the proportion of applications at each active TF value was computed by dividing the number of applications with that value by the total number of applications in the same year and group. These proportions are visualized using color intensity in the heatmap, with darker cells indicating higher proportions. The numeric labels on the heatmap represent the **actual counts** of applications, and bolded numbers indicate the mode of the distribution for each year and group, highlighting the most common number of active TF developers. Proportions are used instead of absolute counts for coloring to account for differences in application development start times across years.

Across all years, the modal number of active TF Developers remains consistently low: one developer for Stable, and zero or one developer for TFDD. This pattern indicates that, regardless of outcome, most applications rely on a very small core of developers.

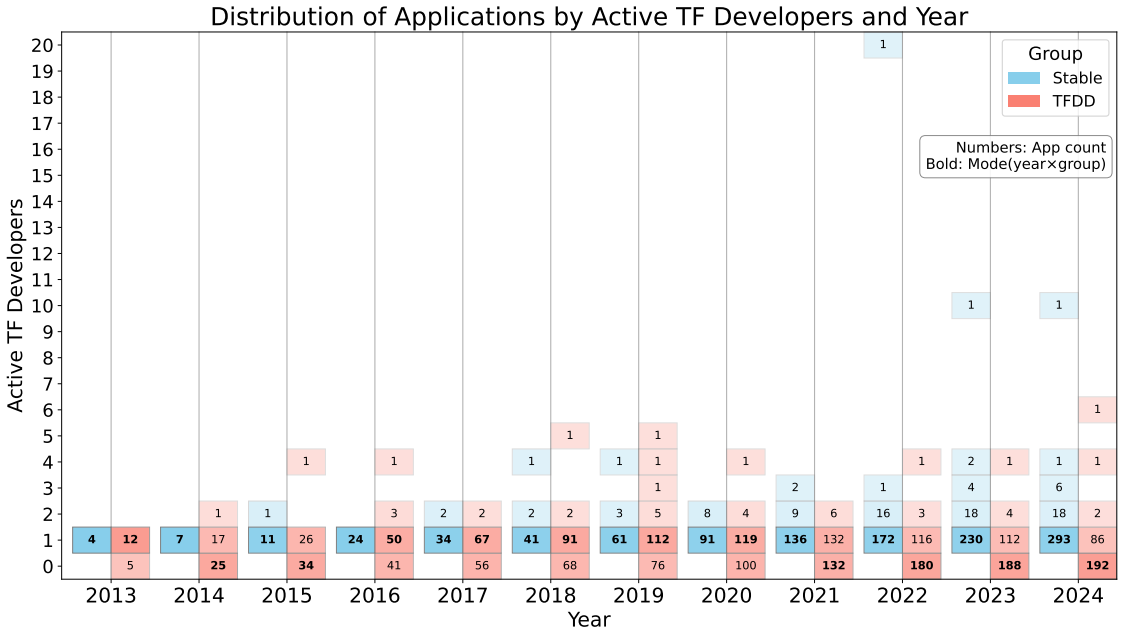


Figure 3: Distribution of the number of TF Developers by year.

For most applications, TFDD events occur at an early stage after the project begins. As shown in Figure 4, over 80% of projects that experience a TFDD have the event within the first three years following the initial commit. Figure 5 shows the time elapsed from the time the app was released on Google Play Store to a first TFDD event (for apps that experienced a TFDD). Our results show in most case the TFDD will take place early after being published and that close to 20% of mobile application will even experience a TFDD event prior to their release on the store (as shown by negative numbers on the plot).

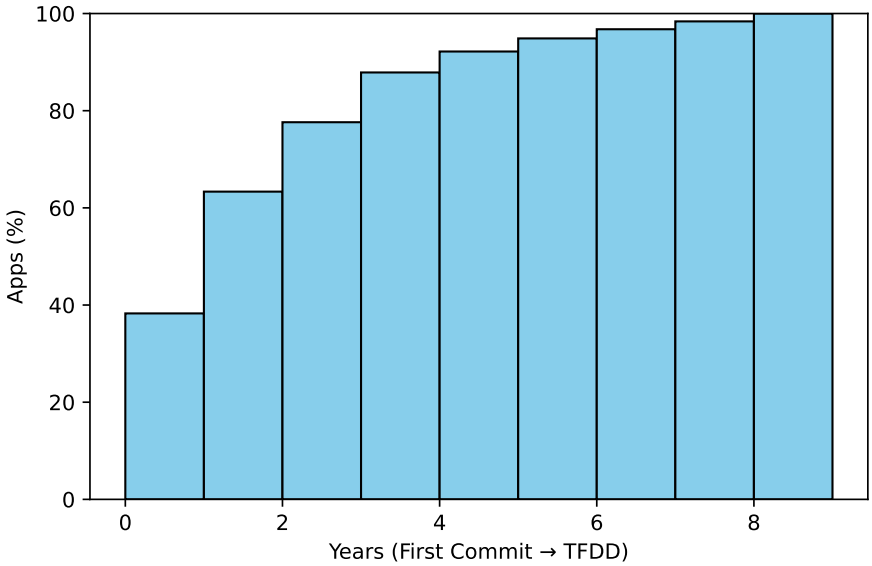


Figure 4: Cumulative proportion of projects experiencing a TFDD within N years of the first commit.

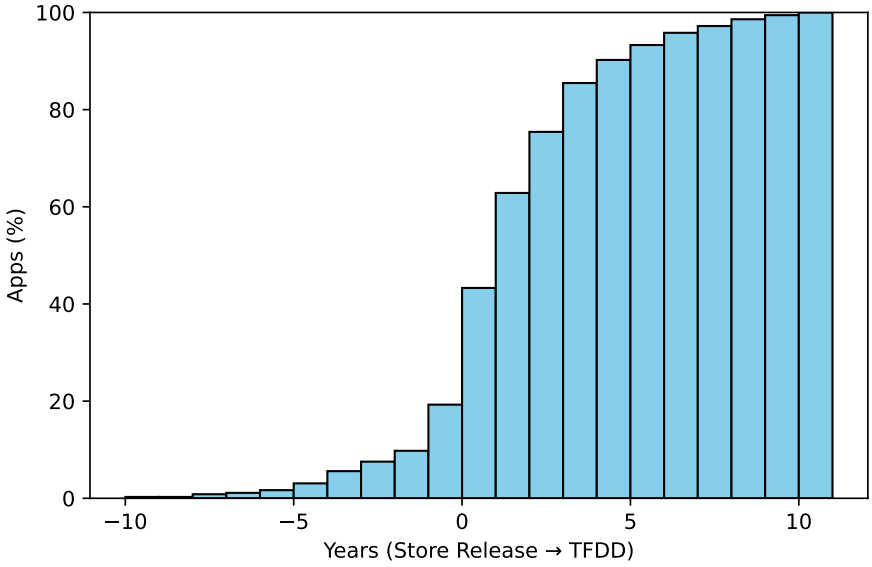


Figure 5: Cumulative proportion of projects that experience a TFDD within N years of their release on Google Play Store.

For Abandoned projects, we investigated the time elapse from the last time an update was pushed to the app on the app marketplace to when a latest TFDD occured. As shown in Figure 6, over 80% of TFDDs happen within the first year when a developer does not update its application on the app store. In a few cases, we also found that a mobile software project had already faced a TFDD, survived, then released the app on the marketplace as shown by the negative numbers in Figure 6.

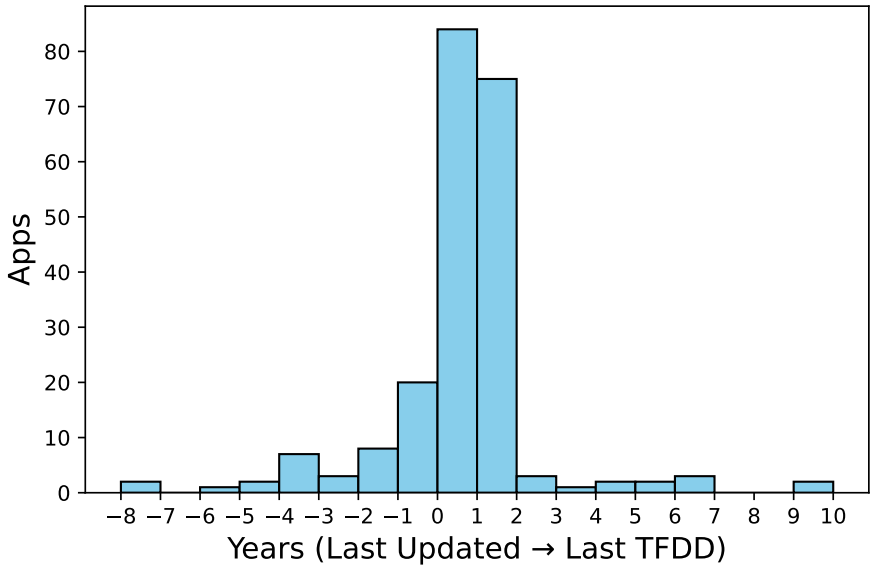


Figure 6: Number of abandoned projects per year from the last update to the occurrence of a TFDD event.

Answer Half of the studied mobile applications experienced a TFDD, with most events occurring within the first three years of development. Our analysis reveals a clear pattern of extreme dependence on a small number of TF Developers, as indicated by the distribution of TF developers and the close temporal relationship between the last update of an app and the occurrence of a TFDD. Only 34.1% of applications were able to recover from a TFDD, highlighting the vulnerability of mobile applications that rely heavily on a limited set of core developers.

4.2 RQ2: What are the characteristics of applications facing TFDD?

Approach. For each group (Stable and TFDD), we analyze the distributions for the app ratings, and the number of installations and user reviews.

Results. As shown in Figure 7, although the Stable group showed a slightly higher median rating (4.06 vs. 3.99), its lower quartiles were distributed at lower values. This suggests greater variability and a higher proportion of poorly rated applications compared with the TFDD group.

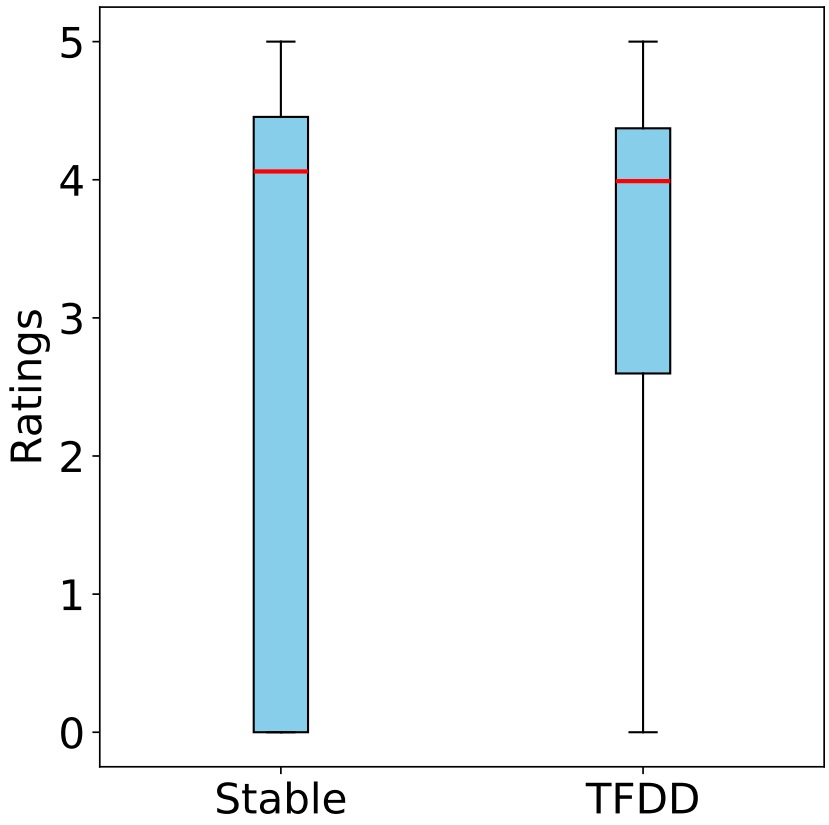


Figure 7: App ratings for Stable and TFDD groups

Distributions of the number of installations are shown in Figure 8. Interestingly, the TFDD group had a higher median number of installations than Stable apps, indicating that applications abandoned by their developers tend to have a larger user base.

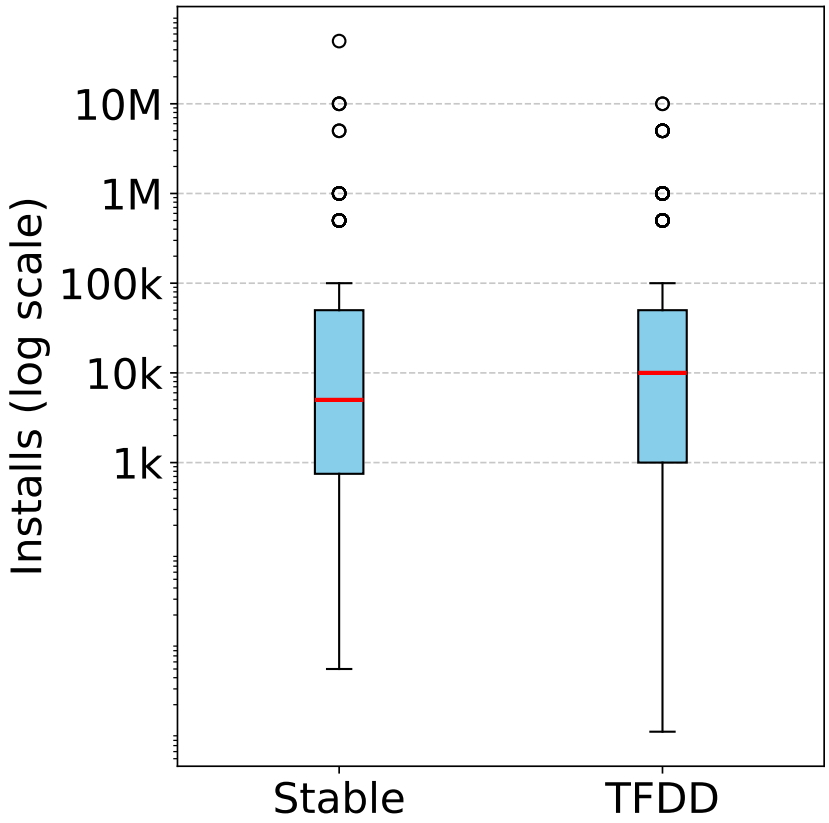


Figure 8: Number of installs for Stable and TFDD groups

Figure 9 illustrates the distribution of review counts. TFDD apps generally had a higher median number of reviews, indicating greater user engagement, whereas Stable apps tended to receive fewer reviews, with a larger proportion of them having only a small number of user reviews.

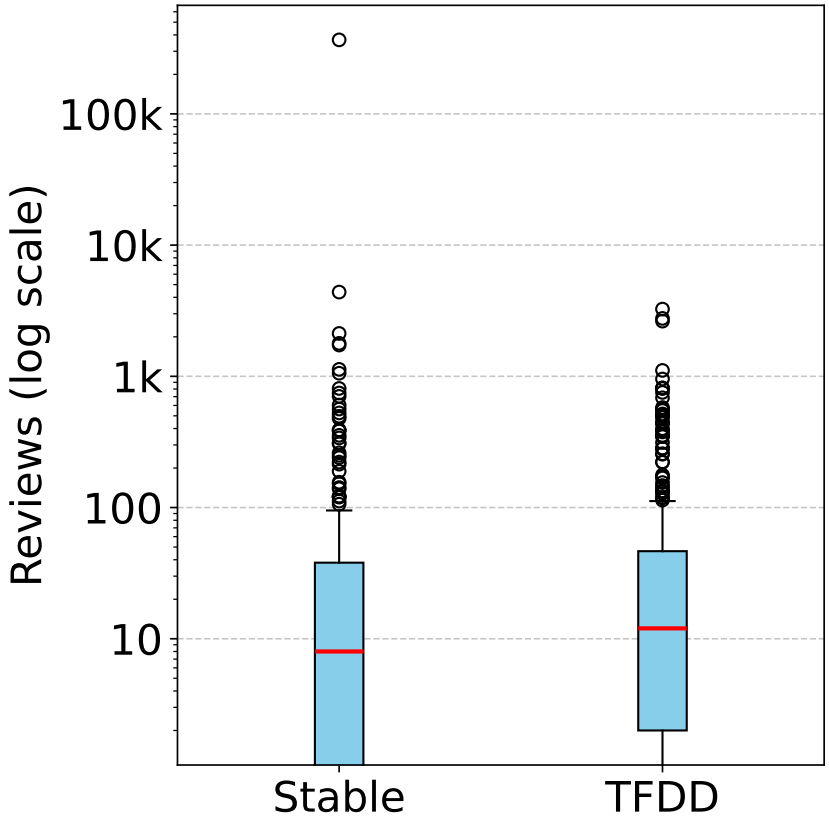


Figure 9: Number of reviews for Stable and TFDD groups

Answer Applications facing TFDD tend to have a larger user base, with higher median numbers of installations and reviews. While Stable applications show slightly higher median ratings, their lower quartiles and smaller user engagement indicate more variability in performance across user metrics. These results suggest that even apps with seemingly high ratings or relatively large user base still has a risk of TFDD. In other words, even apps that appear successful—like those with strong user numbers or positive evaluations —do not necessarily avoid app abandonment.

4.3 RQ3: How do user reviews relate to TFDD events in mobile applications?

Approach. Following the methodology described in Section 3, we use OpenAI's GPT-4o mini model to analyze the content of user reviews and determine 1) the purpose of the review, and 2) the overall sentiment of the user review. Using the LLM, we classified every review from TFDD applications and collected an equal-size random subset of user reviews from Stable applications.

Results. Tables 3 and 4 summarize the results of the LLM-based analysis of user reviews for Stable and TFDD applications.

Regarding review topics, Stable applications exhibit a substantially higher proportion of feedback-oriented reviews (85.03%), whereas TFDD applications show notably higher proportions of feature requests (18.46%) and bug-related reviews (17.18%). This suggests that users of TFDD applications more frequently report unmet feature needs and technical issues.

In terms of sentiment, Stable applications receive a higher proportion of positive reviews (78.58%) and a relatively small share of negative reviews (8.20%). In contrast, TFDD applications display lower overall user satisfaction, with only 64.32% positive reviews and more than twice the proportion of negative reviews (18.82%) compared to Stable applications.

Table 3: Distribution (in %) of review topics across Stable and TFDD applications.

Class	Feedback	Feature	Bug
Stable	85.03	8.68	6.29
TFDD	64.36	18.46	17.18

Table 4: Distribution (in %) of review sentiments across Stable and TFDD applications.

Class	Positive	Neutral	Negative
Stable	78.58	13.23	8.20
TFDD	64.32	16.86	18.82

Answer Our results also show that TFDD apps tend to have a larger proportion of negative reviews, bug reports, and feature requests, highlighting a clear distinction in user feedback compared to Stable applications.

5 Discussion

5.1 Comparison of TFDD Events between Mobile Apps and OSS Projects

Subsection 4.1 provide several insights into characteristics associated with mobile app abandonment and its observable signals. In particular, our analyses demonstrate that app abandonment, measured by TFDD events, is not uncommon: 52.5% of mobile applications published on major app marketplaces, such as the Google Play Store, experienced at least one TFDD event. Among these, 42.6% of apps saw a return of core developers, whereas the remaining 57.4% were completely abandoned. In comparison, Nourry *et al.* [10] analyzed over 36,000 OSS projects and found that 89% experienced TFDD at least once, often in early stages, and only 27% subsequently attracted new core developers. This comparison suggests that while TFDD is relatively common in both domains, the observed differences in outcomes may be attributable to inherent differences in the characteristics of mobile apps and OSS projects.

5.2 Core Developer Dependency and TFDD Risk in Mobile Apps

Mobile applications have consistently relied on very small core development teams since the early days of the Google Play Store. In most years, the majority of apps have only one TF developer (Figure 3). For Abandoned applications, the final update often coincides with the latest TFDD, highlighting the strong dependency on a single or very few core developers. This structural characteristic makes mobile apps highly vulnerable, as their development may lead to app abandonment.

5.3 Comparing Quantitative and Qualitative Signals of Abandonment

At first glance, the findings from RQ2 (Subsection 4.2) and RQ3 (Subsection 4.3) appear to be contradictory. The analysis in RQ2 suggests that applications which experienced TFDD often exhibit strong marketplace performance, including large numbers of installations, a high volume of user reviews, and ratings that were roughly comparable to those of Stable applications. Intuitively, such patterns seem to indicate that an application is successful.

In contrast, the analysis in RQ3 reveals that abandoned applications tend to receive a higher proportion of negative reviews, particularly those reporting bugs, as well as a substantial volume of feature requests. While feature requests are not inherently negative,

their prevalence signal a gap between user expectations and the current capabilities of the application, potentially increasing the maintenance burden placed on developers.

This apparent inconsistency can be largely explained by the different analytical perspectives adopted in RQ2 and RQ3. The metrics used in RQ2 are aggregate, quantitative indicators, such as ratings and installation counts, which primarily reflect historical popularity. In contrast, the review-based analysis in RQ3 captures more fine-grained, qualitative signals that reflect ongoing user dissatisfaction and maintenance-related challenges faced by developers. As a result, reliance on aggregate numerical metrics alone obscures early signals of app abandonment that become apparent only through the content of user feedback.

5.4 Synthesis of the Discussion

Taken together, the above discussion suggest that apparent success based on conventional marketplace metrics does not reliably indicate long-term maintenance. Mobile applications are often highly dependent on a single core developer, making them inherently vulnerable and susceptible to abandonment. Quantitative indicators such as ratings, installation counts, and review volumes may signal historical popularity, but they do not fully capture the underlying risk posed by a single or very small number of core developers.

In contrast, qualitative signals derived from user review content can provide early insights into applications at risk of abandonment. Abandoned apps tend to receive a higher proportion of bug reports and feature requests, reflecting unmet user expectations and potential maintenance burdens. By incorporating such content-based indicators alongside conventional metrics, it is possible to better identify applications at risk, improve app selection or recommendation processes, and ultimately enhance the user experience.

5.5 Practical Implications for Stakeholders Using User Reviews

For researchers, our findings emphasize the need for more rigorous and expressive indicators derived from user review content when studying app abandonment. While our analysis relies on a coarse-grained categorization of reviews, the results suggest that review-based signals capture important aspects of maintenance risk that are not visible through aggregate marketplace metrics. We therefore encourage future work to develop more precise, content-driven measures that better reflect early signs of developer disengagement.

For users, our results suggest that examining recent user reviews can provide valu-

able signals regarding an application's current health. A prevalence of reviews reporting bugs or expressing unmet feature expectations can indicate that an app has already been abandoned or is at risk of abandonment, allowing users to make more informed decisions.

For app store operators, prioritizing qualitative, review-based indicators could support the early identification of at-risk applications. Such an approach may enable more targeted interventions, such as adjusting search rankings or recommendation systems to reduce the prominence of applications showing strong signals of abandonment, thereby benefiting developers, users, and the overall sustainability of the app ecosystem.

5.6 Threats to Validity

5.6.1 Internal Validity

A potential threat to internal validity arises from the characteristics of our dataset. The applications analyzed in this study were drawn exclusively from F-Droid, which hosts open-source Android applications. While this enables direct access to development histories, it introduces a degree of selection bias. Many mobile applications distributed through the Google Play Store are not open source, and applications developed by companies or individual developers often keep their source code private even when they continue to be maintained. Moreover, mobile applications frequently consist primarily of client-side code, for which open-sourcing is not strictly required. As a result, the set of applications available on F-Droid does not fully represent the broader mobile app ecosystem in terms of development practices, organizational support, or maintenance strategies, which can influence the observed relationships between developer activity, user feedback, and app abandonment.

In addition, the size of our dataset is relatively limited, consisting of 706 applications. Although sufficient to observe meaningful patterns, this sample size limits statistical power and increases sensitivity to outliers. Consequently, the reported associations should be interpreted with caution and viewed as indicative rather than exhaustive.

5.6.2 External Validity

A threat to external validity concerns the generalizability of our findings beyond the Android ecosystem. This study focuses exclusively on Android applications, and differences between Android and iOS platforms are likely to influence development practices and abandonment dynamics. Platform governance, review processes, and developer con-

straints differ across ecosystems, shaping update strategies and long-term maintenance behavior.

In particular, prior analyses of mobile app marketplaces have shown that the Apple App Store adopts a more centralized and manually governed review process, with stricter enforcement of platform guidelines and reduced developer autonomy compared to Android-based ecosystems [28]. Such differences in governance and review practices affect development timelines, release frequency, and responses to user feedback, thereby limiting the direct applicability of our findings to iOS applications.

Despite these limitations, some aspects of our findings remain relevant beyond the specific platform analyzed. In particular, the use of review-based signals to identify the risk of development abandonment is not inherently platform-specific and can extend to other software ecosystems in which user feedback is publicly visible. Future work should replicate this analysis using datasets from additional platforms, including iOS, to further assess the generalizability of our findings.

6 Related Work

6.1 Factors Impacting Mobile Application Adoption and Longevity

Several studies have examined factors influencing the adoption and long-term success of mobile applications, primarily from a user-centric or market-oriented perspective.

Ouyang *et al.* [14] conducted an experiment where they created a model to predict/forecast the adoption mobile apps. Their results show that external factors such as marketing and ads and nurturing good reviews helps forecast the popularity trajectory of an app.

Bemmann *et al.* [16] studied the impact of data privacy on the user's decision to adopt a mobile application. Their results showed that users tend to be reluctant to adopt apps which have the ability to execute actions on their behalf. Conversely, some factors such as the expectation of increased productivity made users more likely to adopt an app.

Shen *et al.* [15] studied the impacts that the release strategy can have on mobile applications. Their results showed that apps that are already popular were more likely to maintain or improve their ratings when releasing frequent updates. Additionally, their study showed that the timing of the release also matters and that a well-timed update can reverse a downward trend in ratings. Lastly, their study showed that the purpose of updates is also important and that users tend to respond positively to bug fixing updates.

While these studies provide valuable insights from the user side—such as adoption decisions, privacy perceptions, ratings, and reviews—they do not directly examine how developer-side development activity relates to app abandonment.

6.2 Development Activity and Project Sustainability

Prior studies have investigated how development activity influences sustainability or potential abandonment in OSS ecosystems.

For instance, Foucault *et al.* [29] investigated the impact of developer turnover on software quality in OSS projects. Their results showed that "external turnover" (i.e., developers joining or leaving the project) has a negative impact on the software quality of a software project and the number of bugs.

Nguyen *et al.* [30] analyzed 26,050 GitHub projects to investigate whether projects abandonment when they are no longer actively maintained. Their results show that the risk of severe download loss decreases when a project is consistently maintained, suggesting that maintenance effort plays a critical role in sustaining user interest. Based on these

findings, the authors hypothesize a feedback loop in which declining project relevance or popularity leads to reduced maintenance effort, further accelerating project abandonment.

Structural risks related to developer dependency have been studied using the TF, originally defined by Williams *et al.* [6] as the minimum number of developers whose loss would cause a project to stall or substantially slow down. Building on this original definition, Avelino *et al.* [8] operationalized TF as a scalable and empirically grounded metric, enabling systematic identification of core developers and large-scale analyses of developer dependency in OSS projects. Their work played a key role in establishing TF as a standard measure for assessing structural vulnerability in software projects.

More recently, Avelino *et al.* [9] extended the TF concept by introducing TFDD, which captures periods during which TF developers become inactive. This extension shifted the focus from static assessments of developer dependency to the temporal dynamics of development abandonment, providing a framework for analyzing how the disengagement of key developers threatens project sustainability.

Overall, these studies highlight the role of developer dependency in project outcomes. However, prior work has largely examined these factors either at a structural or conceptual level, or within general OSS ecosystems. In contrast, our study focuses on the practical level of mobile application development, and further relates these events to user-facing metrics, such as ratings and reviews. By explicitly combining developer-centered measures with signals derived from user feedback, this work addresses application abandonment from both the development and user perspectives.

7 Conclusion

In this study, we investigated factors that lead to mobile application abandonment, examining TF and user metrics. We found that half of the studied mobile apps experienced TFDD, and even highly rated and popular apps couldn't avoid TFDD. Lastly, we show that mobile applications that have faced TFDD tend to exhibit a larger proportion of feature requests, bug reports, and overall negative user reviews compared to apps that have not faced.

Taken together, these findings highlight the importance of user reviews as qualitative indicators of app abandonment. Rather than relying solely on quantitative signals such as ratings or popularity, our results suggest that the content of user reviews—including negative content, bug reports, and feature requests—can provide early warning signs of potential abandonment.

This study contributes to a deeper understanding of mobile app abandonment by highlighting the value of qualitative signals derived from user reviews. Based on our findings, researchers can incorporate review content to inform the development of more accurate and robust indicators of potential app abandonment, rather than relying solely on coarse-grained metrics such as ratings or download counts. Such indicators can be useful for both app stores and users: app stores can use them to identify and deprioritize applications likely to be abandoned in search results or recommendations, while users can rely on these indicators to avoid applications at risk of abandonment.

Acknowledgement

As I complete this research, I would like to express my heartfelt gratitude to everyone who has supported and guided me throughout my graduate studies.

First and foremost, I am deeply grateful to Associate Professor Olivier Nourry of the Graduate School of Information Science and Technology, The University of Osaka, for proposing this research topic, providing invaluable guidance throughout the research process, and offering advice on writing and preparing presentation materials. Your mentorship has been instrumental in allowing me to carry out this research successfully.

I would also like to extend my sincere thanks to Professor Yoshiki Higo for his guidance and feedback on the research direction and validity during interim presentations and research meetings, as well as for his support with presentation practice.

I am grateful to Professor Raula Gaikovina Kula for his advice on research direction and English expressions during paper writing.

Associate Professor Makoto Matsushita for his guidance on research methods and precise feedback on presentation materials.

I also sincerely thank Assistant Professor Shinsuke Matsumoto and Professor Shinji Kusumoto for their valuable opinions on the progress of this research.

I would like to thank Ryutaro Inoue and Takuto Kawamoto for their valuable suggestions on research methodology, and Taichi Komura, Tomohito Nagasaki, and Ayaka Yamanaka for their advice during the preparation of this paper.

I am truly thankful to Ms. Mizuho Karube, Ms. Takara Miyazaki for her support in maintaining a comfortable research environment and assisting with administrative tasks, which allowed me to focus fully on my research.

Finally, I would like to express my deepest gratitude to all members of the Higo Laboratory at the Graduate School of Information Science and Technology, The University of Osaka for their guidance, support, and encouragement throughout this research. Your kindness has been a constant source of inspiration and motivation.

Thank you all from the bottom of my heart.

References

[1] Apple App Store. https://www.apple.com/app-store/. Official application marketplace for iOS.

[2] Google Play Store. https://play.google.com/store. Official Android application marketplace.

[3] Statista. Number of apps available in leading app stores. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app 2025. Accessed via Statista; subscription required.

[4] 42matters / AppBrain. Number of apps available for download. https://aboutchromebooks.com/how-many-apps-are-available-for-download/, 2025. Apple App Store approximately 2.02 million apps, Google Play Store approximately 1.60 million apps.

[5] Coatue. Chart of the day: Agentic coding is accelerating app releases. Industry commentary, 2026.

[6] L. Williams and R. Kessler. *Pair Programming Illuminated*. Addison-Wesley, 2003.

[7] F. Ricca and A. Marchetto. Are heroes common in floss projects? In *Proc. 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, number article no.55, pages 1–4. Association for Computing Machinery, 2010.

[8] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A novel approach for estimating truck factors. In *Proceedings of the 24th IEEE International Conference on Program Comprehension (ICPC)*, pages 1–10, 2016.

[9] Guilherme Avelino, Eleni Constantinou, Marco Túlio Valente, and Alexander Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–12. IEEE, 2019.

[10] Olivier Nourry, Masanari Kondo, Shinobu Saito, Yukako Iimura, Naoyasu Ubayashi, and Yasutaka Kamei. Myth: The loss of core developers is a critical issue for oss communities. *arXiv preprint arXiv:2412.00313*, 2024.

[11] Fabio Ferreira, Luciana Lourdes Silva, and Marco Tulio Valente. Turnover in open-source projects: The case of core developers. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, page 447–456. Association for Computing Machinery, 2020.

[12] Fabio Calefato, Marco Aurélio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. Will you come back to contribute? investigating the inactivity of oss core developers in github. *Empirical Softw. Engg.*, 27, 2022.

[13] Iso 9241-11: Ergonomics of human-system interaction — part 11: Usability: Definitions and concepts, 2018.

[14] Yi Ouyang, Bin Guo, Tong Guo, Longbing Cao, and Zhiwen Yu. Modeling and forecasting the popularity evolution of mobile apps: A multivariate hawkes process approach. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2, 2018.

[15] Sheng Shen, Xuan Lu, Ziniu Hu, and Xuanzhe Liu. Towards release strategy optimization for apps in google play. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware*. Association for Computing Machinery, 2017.

[16] Florian Bemmann and Sven Mayer. The impact of data privacy on users' smartphone app adoption decisions. *Proceedings of the ACM on Human-Computer Interaction*, 8, 2024.

[17] Simo Sigg, Eero Lagerspetz, Esa Peltonen, Petteri Nurmi, and Sasu Tarkoma. Sovereignty of the apps: There's more to relevance than downloads. In *arXiv preprint arXiv:1611.10161*, 2016.

[18] M. Ferreira, M.T. Valente, and K. Ferreira. A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 207–217, 2017.

[19] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, and V. Kovalenko. Bus factor in practice. In *Proc. 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 97–106. Association for Computing Machinery, 2022.

[20] Tobias Fritz, Jeff Ou, Gail C. Murphy, and Emily Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *32nd International Conference*

on Software Engineering (ICSE), pages 385–394, 2010.

[21] Tobias Fritz, Gail C. Murphy, Emily Murphy-Hill, Jeff Ou, and Emily Hill. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology*, 23(2):14:1–14:42, 2014.

[22] kevinah95 cbaenziger lkwg82 gavelino, mtov. Truck-factor. `https://github.com/aserg-ufmg/Truck-Factor/graphs/contributors`, 2015. Accessed: 2025-10-02.

[23] Anonymous. F-droid tabler: Web dashboard interface for f-droid repository [online]. Available: `https://fdroid.tabler.dev/`. Accessed: Oct. 18, 2025.

[24] google-play-scraper. `https://pypi.org/project/google-play-scraper/`. Python library for scraping Google Play Store metadata and reviews.

[25] Nikkei Shimbun. 名称変更だけではない、android マーケットの変身 「無審査」だった検査体制も徐々に改善へ (english translation: Not just a name change: The transformation of the android market, gradual improvement of the previously "no-review" inspection system). `https://www.nikkei.com/article/DGXNASFK1803H_Y2A510C1000000/`, May 2012. Accessed February 6, 2026; membership required.

[26] OpenAI. GPT-4. `https://openai.com/ja-JP/index/gpt-4/`. Accessed 2026.

[27] OpenAI. OpenAI API. `https://openai.com/ja-JP/api/`. Accessed 2026.

[28] Competition and Markets Authority. Mobile ecosystems market study: Interim report. Government market study, 2021. UK government report analyzing app store governance, review processes, and developer constraints.

[29] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, page 829–841. Association for Computing Machinery, 2015.

[30] Emily Nguyen. Do all software projects die when not maintained? analyzing developer maintenance to predict oss usage. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 2195–2197. Association for Computing Machinery, 2023.