UCL DEPARTMENT OF COMPUTER SCIENCE

Research Note
RN/15/07

Causal Impact Analysis Applied to App Releases in Google Play and Windows Phone Store

December 16, 2015

William Martin

Federica Sarro

Mark Harman

Abstract

App developers would like to know the characteristics of app releases that achieve high impact. To address this, we mined the most consistently popular Google Play and Windows Phone apps, once per week, over a period of 12 months. In total we collected 3,187 releases, from which we identified 1,547 for which there was adequate prior and posterior time series data to facilitate causal impact assessment, analysing the properties that distinguish impactful and non-impactful releases. We find that 40% of target releases impacted performance in the Google store and 55% of target releases impacted performance in the Windows store. We find evidence that more mentions of features and fewer mentions of bug fixing can increase the chance for a release to be impactful, and to improve rating.

1 Introduction

The motivation for our study derives from the need for developers to understand the impact of their releases [9, 12, 17, 18, 21, 28, 36]. The high release frequency in app stores, and the correlation between performance metrics and revenue that accrues to developers from those releases, can lead to the adoption of rapid release strategies. Such strategies can offer significant benefits to both developers and end users [28], but high code churn in releases can correlate with decreased ratings [17]. Releases can happen for a number of reasons such as updating advertisement libraries [18], or stimulating performance [9], in addition to more traditional bug fixes, feature additions and interface improvements. Developers may even find certain days of the week can stimulate performance more than others [12, 21]. McIlroy et al. [36] studied update frequencies over 7 weeks in the Google Play store, finding that 1% of apps received updates more than once per week, and 14% of the studied apps were updated in a two-week period. They found that rating was not affected by update frequency.

In this study, we record time-series information about apps, and identify how release frequency can affect an app's performance as measured by rating, popularity and number of user reviews. We identify the set of 'impactful' releases, that statistical evidence suggests caused a significant change in one of their app's performance metrics, compared with a baseline set of non-releasing apps. We then analyse the characteristics of these highly impactful app releases, in order to understand the properties under developer control which could lead to releases impacting performance.

We use Causal Impact Analysis [6], a form of causal inference [22, 33], in order to identify those releases that are significantly likely ($p \leq 0.01$) to have changed an app's rating, download rank or user reviewing frequency. Causal inference is primarily used in economic forecasting, for measuring or predicting the effect of an event on time-series data, but it has also seen recent use for software defect prediction [10, 11, 53].

We use information retrieval to investigate the top terms and topics that occur in the release text of highly impactful releases, and also investigate the overall effects of release frequency and release interval on app performance. Our results provide actionable findings for developers in Google Play and Windows Phone stores, and are also likely to apply to other app stores.

To facilitate this study, we collected data over a 52 week time period concerning the most popular apps in the Google Play and Windows Phone app stores. We set the time interval between which we collected snapshots of the two app stores to one week; since only 1% of apps release more frequently than once per week, and these have been studied in detail by McIlroy et al. [36], our collection interval seems representative of the majority of releasing behaviours of the studied apps. We are interested in investigating evidence that release behaviour has an impact on app performance, in order to determine whether developers' release behaviour is justified. Hitherto, it is unknown whether particular strategies have an impact on subsequent app performance, making findings that characterise these impacts important for software engineers.

Our contributions are as follows:

i) We find that 40% of target releases are impactful in Google and 55% are impactful in Windows.

ii) We find that impactful releases had proportionally fewer mentions of (bug, fix) in their release text than non-impactful releases: 33% to 38% in the Google store and 44% to 48% in the Windows store.

iii) We find that releases that positively impacted rating had proportionally fewer mentions of (bug, fix) than those that negatively impacted rating: 29% to 33% in the Google store and 38% to 43% in Windows store, and more mentions of (new, feature): 30% to 25% in Google store and 55% to 39% in Windows store.

iv) Our results indicate that Causal Impact Analysis is useful for identifying impactful releases for further analysis.

2 App Metrics Collected

The questions we ask involve recording changes to apps and changes in their performance. In order to assess performance, the following app-level metrics are used:

(R) Rating: The average of user ratings made for the app since its first release on the store.

(D) Download rank: Indicates the app's popularity, although specifics of the calculation of download rank vary between app stores and are not released to the public. The rank is in descending order. That is, it increases as the popularity of the app decreases: from a developer's perspective, the lower the download rank, the better. We report download rank as-is in box plots, but in tables indicating positive or negative gains for metrics, we report decreases in the download rank as positive (and increases as negative), since decreases correspond to benefit to the developer.

(N) Number of ratings: The total number of ratings that the app has received.

(NW) Number of ratings per week: The number of ratings that the app has received since the previous snapshot, which is taken a week earlier.

The following metrics are under developer control:

(P) Price: The amount a user pays for the app in GBP in order to download it. This value does not take into account in-app-purchases and subscription fees, thus it is the 'up front' price of the app.

Version identifier: An app release occurs when an app snapshot has a different version identifier for the released app compared to the previous snapshot.

(RT) Release text: The app's *description* (if changed) and the '*what's new*' section of the app store for that app (if changed).

3 Research Questions

This section explains the questions posed in our study and how we approach answering them.

RQ1: Do app metrics change over time?

Before performing detailed analysis of the changes over time, we first set a baseline by establishing whether app performance metrics change between snapshots. Using the metrics R, D, N and NW as defined in Section 2, we compute their standard deviation over 52 weeks for each app, and draw them on box plots. This enables us to establish whether metrics change over time, and to what extent this occurs. If the metrics do change over time, it motivates further analysis of events that could cause changes over time, specifically releases.

RQ2: Do release statistics have a correlation with app performance?

We measure whether app performance is affected by the number of app releases by measuring the correlation between performance metrics and the number of releases in 52 weeks, as well as the change in metrics over 52 weeks.

RQ2.1: Does the number of releases have a high correlation with app performance? We perform correlation analysis between the number of releases of each app and their current value for the metrics R, D and N. We do not use NW because this number is set on a per-week basis, but instead use the change in R, D and N from the first snapshot to the last, denoted ΔR, ΔD and ΔN respectively.

RQ2.2: Does the median time interval between releases have a correlation with app performance? We perform correlation analysis between the median interval between multiple releases of each app and the metrics used in RQ2.1. For this, only apps which have releases in the time period are used.

RQ3: Do releases impact app performance?

There are two limitations to correlation analysis. Firstly, correlation analysis seeks an overall statistical effect in which a large number of apps and their releases participate. However, it may also be interesting, from the developers' point of view, to identify specific releases that have an atypically high impact, compared to the 'background' behaviour and characteristics of the app store as a whole; general correlation analysis is not well-suited to this more specific question. Secondly, of course, as is well-known any correlation observed does not necessarily imply the presence of a cause (correlation is not causation). Therefore, even were we to find strong correlations, this would not, of itself, help us to identify causes. This motivates our use of causal impact analysis. We apply the causal impact analysis on each target release to see if it caused a significant change in any of the app-level performance metrics R, D, N or NW defined in Section 2. Causal impact analysis is described in Section 4.3.

RQ3.1: What proportion of releases impact app performance? We compute the proportion of apps whose releases have affected performance, and the proportion of overall releases.

RQ3.2: How does the causal control set size affect results? Causal impact analysis requires a control set (in our case, a set of apps that have zero releases in the period studied). As the set of potential members of the control set is different in size between Google and Windows, we carry out experiments to assess how much control set differences could influence the results. Our approach is similar to the experiment using different control sets in the study by Brodersen et al. [6]. We compute the causal impact analysis results for each metric for a sample of 100 target releases in the Windows dataset, using control sets of size 100, 200 and 397, the smaller of which are randomly sampled from the maximum possible set of 397 non-releasing apps.

RQ4: What characterises impactful releases?

We use the causal impact analysis results from RQ3 to analyse impactful and non-impactful releases.

RQ4.1: What are the most prevalent terms in releases? We pre-process the release text from all releases in a given store as described in Section 4.4, then identify the 'top terms' (most prevalent) for each set of releases using two methods: **TF.IDF** [34] and **Topic Modelling** [4].

We train both methods on the release text corpus, treating each instance of release text as a document. Each store is treated separately for training and evaluation, to prevent combining store-specific vocabularies. For both methods, we sum the resultant scores (probabilities) for terms (topics) over each set of releases. We restrict ourselves to only the top three terms to avoid over specialisation.

The topic model is trained using 20 topics in each case. We chose 20 topics for two reasons: i) the number of documents in each corpus is between 1000 and 2000, each consisting of tens to hundreds of words, which is by no means a *large* corpus; ii) we wish to generalise, to avoid training topics that are relevant to certain apps or releases. The choice of 20 topics allows us to generalise and easily inspect each of the trained topics, without much risk of training a topic that is overly specific to an app or release.

RQ4.2: How often do top terms and topics occur in each set of releases? We compute the counts in each set of releases that contain top terms from TF.IDF and topic modelling, as identified from RQ4.1. We apply a bag-of-words model, which ignores the ordering of the words in each document. This eliminates the need to check for multiple forms of text that is discussing the same thing.

4 Methodology

This section describes the methods used in our study.

4.1 Data Mining

The download rank is unavailable for apps outside the 'top' free or paid lists for both Windows Phone and

Google Play app stores. For Google Play, this information is available for only the most popular 540 free and 540 paid apps, while for Windows Phone, download information is available for only the most popular 1000 free and 1000 paid apps.

We therefore mined apps from the Google Play and Windows Phone stores (on a weekly basis), recording the most popular 540 free and 540 paid apps from Google, and top 1000 free and 1000 paid apps from Windows. The time period we considered was from July 2014 to July 2015, giving 52 snapshots for each of Windows Phone and Google Play. In order to focus solely on the consistently popular apps, we computed the intersection of 52 snapshots. This resulted in 307 consistently popular (i.e. always within the top 540) Google apps, and 726 consistently popular (i.e. always within the top 1000) Windows apps. These 1,033 apps are those studied in this paper.

Our conclusions thus concern the effect of releases on the most consistently popular apps over a period of the year (July 2014 to July 2015), and care will be required before the results could be extended to apps and their releases more generally, due to the App Sampling Problem [35]. Nevertheless, we believe that conclusions about the characteristics of impactful releases for the most consistently popular apps will yield interesting and actionable findings for developers, because consistently popular apps are an inherently developer- attractive and interesting subset of app stores.

We extract app metrics from each of the 1,033 apps including price, rating, download rank, number of ratings, description, what's new and version. In the case of Google Play, the app store reports rounded app ratings (rounded to 1 decimal place), thereby creating a potential source of imprecision, which we would like to overcome. To improve comparability between the two app stores, we would also like to ensure that ratings are computed to the same precision. Therefore, we recalculate the (more precise) Google Play average ratings, using the extracted numbers of ratings in each of the five star ratings (from 1-5).

4.2 Explanation of the Frequentist Inferential Statistical Analysis Techniques Used

We use correlation analysis in order to understand correlations between the observed app metrics and both the quantity and interval of their releases.
However, since it is well known that 'correlation does not imply causation', we further investigate the impact of releases using causal impact analysis (explained in more detail in Section 4.3). We follow up the causal impact analysis with a nonparametric inferential statistical analysis, to provide further evidence as to the relative likelihoods that each of the app properties observed plays a role in the causal impacts detected by causal impact analysis.

This subsection explains the role played, in our overall analysis, by traditional frequentist inferential statistical techniques (with which software engineers are most likely to be already familiar), while the following subsection explains causal impact analysis (which is comparatively less widely used in the domain of software engineering [6, 11, 10, 33, 53]).

In RQ2 and RQ4 we use Pearson and Spearman statistical correlation tests. Pearson introduced the measurement of linear correlation [41], while Spearman subsequently extended Pearson's work to include rank-based correlation [46]. Pearson's technique measures the linear correlation between two paired vectors of data, while Spearman's technique extends this to rank-based correlation.

Each correlation metric reports a rho value and a p value. The p value denotes the probability that a rho value is different to zero (no correlation). A rho value of 1 indicates perfect correlation, while -1 indicates perfect inverse correlation, and 0 indicates no correlation. Values between 0 and 1 (-1) indicate the degree of correlation (inverse correlation, respectively) present.

In RQ3 and RQ4 we also compare distributions using a two-tailed unpaired non-parametric Wilcoxon test [51], that tests against the Null-hypothesis that the result sets are sampled from the same distribution. We also compare the result sets using Vargha and Delaney's $\hat{A}_{12}$ effect size comparison test [50], which results in a value between 0 and 1, that tells us the likelihood that one measure will yield a greater value than the other.

In our case, we apply these inferential statistical tests to examine differences in properties between the sets of releases that have demonstrated a causal impact (using causal impact analysis) and those which have not, as well as between the sets of releases that positively impacted rating and those that negatively impacted rating. The p value is the probability that we would observe the difference in median values we find, given that there is, in fact, no difference in the two distributions from which the releases are drawn. It is a conditional probability, usually used to reject the null hypothesis (that there is no difference).

However, in our case, a prior causal impact analysis has revealed that there is *some* causal difference between the two sets, yet it remains unknown what this cause is. Causal impact analysis does not fully overcome the problem that 'correlation is not causation', but it does provide greater evidence for causal impact (in our case causal impact of a release). Therefore, the p value can be interpreted as an indication of the relative *likelihood* that the property tested is one of the influencing factors in the causal impact already detected by causal impact analysis. As such, we are not looking for statistical significance (seeking a p below some arbitrary predefined threshold). Instead, we use the relative probabilities of rejecting the null-hypothesis as an indicator of the relative likelihood that each of the properties tested plays a role in the causal impact detected.

Since we are computing multiple p values, the reader might expect some kind of correction, such as a Bonferroni or Benjamini-Hochberg [3] correction for multiple statistical testing at the (traditionally popular) 0.05 probability level (corresponding to the 95% confidence interval). However, since we are *not* using p values to test for significance; should a p value lie above this (corrected) threshold, then this *does not* necessarily indicate that the property does not contribute to the observed causal impact. Quite the contrary; since we have already observed that there exists a causal impact, then the property that exhibits the lowest p value remains that with the highest probability of having *some* influence on the causal impact, amongst those properties assessed using inferential statistics.

4.3 Causal Impact Analysis

We apply the Causal Impact Analysis method [6] using the Google `CausalImpact` framework [5]. Each metric (R, D, N, NW) and each release, requires an individual experiment, as the method works each time on a single data vector.

The Causal Impact Analysis method we use in RQ3 and RQ4 trains a Bayesian Structural Time-Series model [22, 45] on the data vector for each **target release**, using a set of unaffected data vectors known as the **control set**. This allows the model to make a prediction of the data vector in the posterior time period using a set of regression coefficients and biases, accounting for local, cyclic and global variations.

The method computes the probability that the observed data vector post-release could have occurred by comparing it with the counterfactual prediction. A low value ($p \leq 0.01$) indicates that the performance metric changed significantly; see for example Fig. 1, which shows that the rating of 'Carp Fishing Simulator' deviates significantly from the predicted vector. Of course, the degree to which we can assume causality relies on the strength of our causal assumptions, that the control set is unaffected by the release, and the relationship of the control to the released app is unchanged.

We use the method to identify **impactful releases** (defined below), and look for trends amongst them which may hold true, using inferential statistical analysis as explained in Section 4.2. Our causal impact analysis can only tell us that there is evidence that a release has had an impact on the metrics we collect, and our subsequent inferential statistical analysis can only point to the relative likelihoods that the properties we investigate play a role in this causal impact.

In any and all causal impact analyses, it is of course impossible to identify external uninvestigated properties, that might also have played a role in the causal impacts observed. In the case of app stores, our current analysis cannot, for example, account for the possible impacts of advertising campaigns, timed to coincide with the new release. While such campaigns might plausibly contribute to the causal impact of the release, we have no information about such campaigns, and therefore cannot take it into account.

We use the following data to answer RQ3 and RQ4:
Control set: apps that have no releases in the 12 month time period: for Google this set is 97 apps, and for Windows this is 397 apps. We compare different control set sizes in RQ3.2 in order to establish whether the choice of control set affects the results.
Target releases: releases that occurred at least 3 weeks after the previous release of the same app (or start of the time-series), and that occur at least 3 weeks before the next release (or end of the time-series). This ensures sufficient data availability to accurately train causal impact analysis.
Impactful release: a release for which one of the performance metrics (R, D, N and NW) significantly deviated from the counterfactual prediction. We consider a significant deviation to be one in which the probability of predicting the observed is ≤ 0.01. This is a cautious choice of probability (corresponding to the 99% confidence interval) to reduce the likelihood of raising false alarms, which subsequently would turn out not to be truly impactful after all. By setting our threshold to be 0.01, we have a 0.01 probability of claiming an impact where one does not exist, and therefore expect roughly 1% false positive rate.

4.4 Information Retrieval Techniques

To answer RQ4, we perform information retrieval analysis on release text, using both TF.IDF and Topic Modelling. We use two different techniques to increase the confidence with which we can identify the top terms that occur in impactful release text.
Filtering: Text is cast to lower case and filtered for punctuation and stopwords, using the English language `stopwords` from the Python NLTK data package[1].
Lemmatisation: Each word is processed by the Python NLTK `WordNetLemmatizer`, in order to be transformed into its 'lemma form', to homogenise singular/plural, gerund endings and other non-germane grammatical details.
TF.IDF: TF.IDF [34] finds 'top terms' in release text: each term in each document is given a score of TF (Term Frequency) multiplied by the IDF (Inverse Document Frequency). The IDF is equal to the log of the size of the corpus divided by the number of documents in which the word occurs.
Topic Modelling: Topic modelling [4] finds top topics in release text. Topic modelling is a generative probabilistic technique that trains a graphical model on a set of unstructured textual documents, under the assumption that they are generated from a set of latent topics.

5 Results

This section answers the questions posed in Section 3.
RQ1: Do app metrics change over time? We can see from the box plots in Fig. 2 that the metrics (R)ating, (D)ownloads, (N)umber of reviews and (NW) number of reviews per week do, indeed, change over time, because their median standard deviation is always positive.

However, Fig. 2 reveals that not all metrics vary so greatly: the (R)ating metric exhibits the least variation (median standard deviation < 0.05 for both Google and Windows). This is a potentially useful baseline finding, because it means that a high-impact release (that does affect rating), has a chance to 'stand out against the crowd'.

[1] `nltk.corpus.stopwords.words('english')`

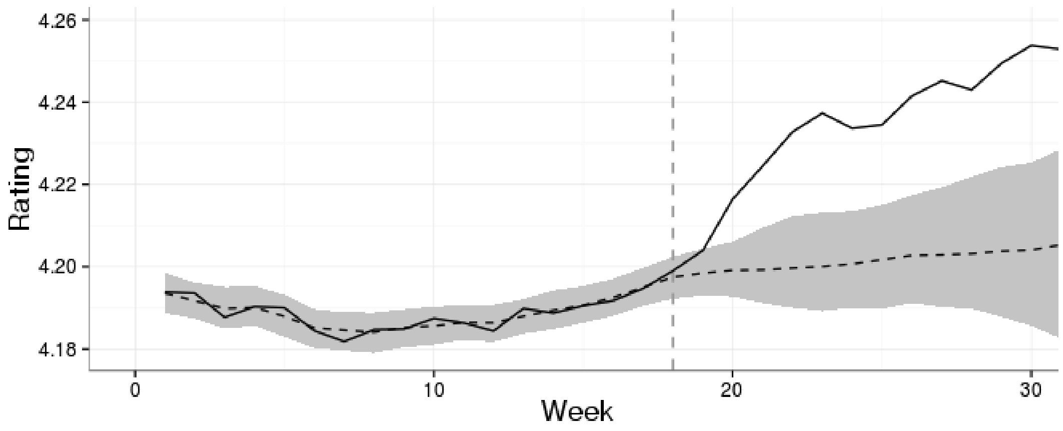

Figure 1: **Causal impact analysis** for the rating of Carp Fishing Simulator in Google Play. The shaded region surrounding the dotted trend line represents the confidence interval for counterfactual prediction. The vertical dotted line denotes the release point, after which, the actual behaviour (solid line) clearly moves outside the confidence interval.

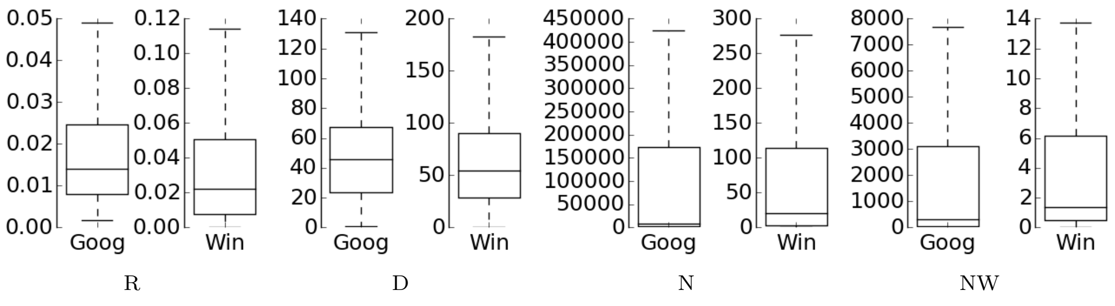

Figure 2: RQ1: Standard deviation box plots for (R)ating, (D)ownload rank, (N)umber of ratings and (NW) number of ratings per week.

Developers are likely to care about such releases, since ratings are important, and there is some evidence that they impact upon popularity, and thereby revenue [20].

RQ2: Do release statistics have a correlation with app performance? Having established a baseline, we now measure the correlations between the number and interval of releases in 52 weeks, and performance metrics.

RQ2.1: Does the number of releases have a high correlation with app performance? Table 1 presents the results of correlation analysis between release frequency and app metrics for Google and Windows app stores. We only report correlation coefficients (the rho values) that are deemed significant ($p <= 0.05$), i.e., where there is sufficient evidence that rho $\neq 0$.

The results from our Google dataset in Table 1 are sparse, indicating only 1 significant correlation between the number of releases and ΔN, the change in the number of reviews. Even this correlation, although significant, is still weak (rho = 0.13) so we conclude that there is little evidence for correlation between release frequency and app metrics in the Google app store. A greater number of significant correlations was observed for the Windows app store (Table 1). However, the corresponding correlation coefficients (rho values) remain low, providing evidence only for a mild correlation between release frequency and the change in the number of reviews per week (ΔN).

We therefore conclude that there is no strong overall correlation between release frequency and the app metrics we collect for either app store, but there is evidence for a mild correlation between release frequency and number of reviews in 52 weeks in the Windows store.

RQ2.2: Does the median time interval between releases have a correlation with app performance? Table 2 presents the results of correlation analysis between release interval and app metrics for the Google and Windows app stores. As these results revealed, there is little evidence for any strong correlation between the median inter-release time period and the app metrics we collect. Our findings corroborate and extend the recent findings by McIlroy et al. [36], who reported the rating was unaffected by release frequency in the Google app store. This is interesting because there is evidence that app developers release more frequently when an app is performing poorly [9]; our results indicate that this, perhaps rather desperate behaviour, is unproductive.

RQ3: Do releases impact app performance? The results from RQ1 and RQ2 have established that app performance metrics do vary over releases, but that the number of releases and time intervals between releases are not important factors in determining these performance changes. This makes causal impact analysis potentially attractive to developers. With it, a developer can seek to identify the set of specific releases that had a higher performance impact, using evidence for significant changes in post-release performance compared with the set of non-releasing apps.
This is the analysis to which we now turn in RQ3.

RQ3.1: What proportion of releases impact app performance? Table 3 presents overall summary statistics for the results of causal impact analysis. The row labelled 'Apps' indicates the number of apps summarised in each of the two sub tables (307 Google apps and 726 apps for Windows). This is the total number of apps which remain consistently popular over all 52 weeks studied. The total number of releases reports the number of app releases over the 52 weeks studied, while the 'target releases' denotes the subset of releases for which there is sufficient prior and posterior information available to support causal impact analysis, in terms of counterfactual posterior predictions, based on prior observations. Those releases that occur near the beginning or end of the time period will therefore not have sufficient information available, and so the causal impact cannot be studied; hence we select a subset of releases that must also belong in the range of weeks [4, 49], out of a possible [1, 52]. Of these target releases, some are impactful and some are not according to causal impact analysis. As Table 3 reveals, we found 39.9% of the target releases in the Google store and 55.1% in the Windows store to be impactful.

The remainder of Table 3 reports the observed change in performance metrics for impactful releases, thereby identifying candidate causes of these impacts. For each performance metric change, we report the total number of releases that exhibited an impactful change in the associated metric and the percentage (of all app releases) that exhibited the change. We further subdivide this total into those which are considered positive and those which are considered negative from the developers' perspective.

From the 39.9% of impactful Google releases, approximately a third (31.9%) impacted more than one performance metric. The releases of most potential interest to developers are those that impact rating and download rank (since these are most closely coupled to revenue), of which there were 32 impactful releases.

In the Windows dataset there were a higher proportion of impactful releases: 55.1% were impactful in some performance metric, and of these impactful releases, approximately half (49.7%) had an impact on multiple metrics. There were 90 releases in the Windows dataset that impacted rating and download rank, and 11 that impacted rating, download rank and number of ratings per week.

These results support the hypothesis that there is a subset of releases that cause significant changes to their app's performance in the store.

RQ3.2: How does the causal control set size affect results? Table 4 reports the effect of choosing different control set sizes, from among those apps which did not undergo any releases during the time period studied. The Table 4 results show that very similar findings are observed for impactful releases (with respect to each performance metric), irrespective of the choice of control set.

RQ4: What characterises impactful releases? The finding from RQ3, tells us that there are impactful releases in both app stores, but it cannot identify the causes, merely that there has been an impact in post-release performance. We now turn to analyse candidate causes, and investigate the relative probability that each

Table 1: RQ2.1: Significant ($p \leq 0.05$) correlations between number of releases and (R)ating, (D)ownload rank and (N)umber of ratings at the end of the 52 week time period, as well as the change in these metrics from first to last week.

Method	R	ΔR	D	ΔD	N	ΔN
Spearman	-	-	-	-	-	0.13
Pearson	-	-	-	-	-	-

Google

Method	R	ΔR	D	ΔD	N	ΔN
Spearman	0.20	-	-0.17	-	0.32	0.42
Pearson	0.16	-	-0.16	-0.09	0.27	0.34

Windows

Table 2: RQ2.2: Significant ($p \leq 0.05$) correlations between release interval and (R)ating, (D)ownload rank and (N)umber of ratings at the end of the 52 week time period, as well as the change in these metrics from first to last week.

Method	R	ΔR	D	ΔD	N	ΔN
Spearman	-	-	-	-	-0.15	-0.19
Pearson	-	-	0.16	-	-	-

Google

Method	R	ΔR	D	ΔD	N	ΔN
Spearman	-	-	0.12	0.13	-	-
Pearson	-	-	0.13	-	-	-

Windows

Table 3: RQ3.1: Causal impact analysis results.

Details of target releases (percentages reported over 754 target releases)

Type	Total (of target)	Apps	307
Non-impactful	453 (60.1%)	Total releases	1,570
Impactful	301 (39.9%)	Target releases	754

Details of impactful releases (percentages reported over 301 impactful releases)

Metric	Total (of impactful)	+ve	-ve
R	152 (50.5%)	67 (22.3%)	85 (28.2%)
D	130 (43.2%)	48 (15.9%)	82 (27.2%)
N	54 (17.9%)	54 (17.9%)	0
NW	84 (29.9%)	52 (17.3%)	32 (10.6%)
R ∩ D	32 (10.6%)	9 (3.0%)	12 (4.0%)
R ∩ D ∩ NW	7 (2.3%)	2 (0.7%)	1 (0.3%)

Google

Details of target releases (percentages reported over 793 target releases)

Type	Total (of target)	Apps	726
Non-impactful	356 (44.9%)	Total releases	1,617
Impactful	437 (55.1%)	Target releases	793

Details of impactful releases (percentages reported over 437 impactful releases)

Metric	Total (of impactful)	+ve	-ve
R	228 (52.2%)	90 (20.6%)	138 (31.6%)
D	207 (47.4%)	93 (21.3%)	114 (26.1%)
N	267 (61.1%)	267 (61.1%)	0
NW	48 (11.0%)	27 (6.2%)	21 (4.8%)
R ∩ D	90 (20.6%)	15 (3.4%)	36 (8.2%)
R ∩ D ∩ NW	11 (2.5%)	2 (0.5%)	5 (1.1%)

Windows

Table 4: RQ3.2: Results indicating minimal effects from different control set choices using a test set of 100 target releases.

	Control Set		
Result	100	200	397
R results	45	43	39
D results	40	42	41
N results	36	40	40
NW results	13	13	14

has played a significant role in causing the impacts we have observed.

RQ4.1: What are the most prevalent terms in releases? Table 5 reports the results of information retrieval, using TF.IDF and the topic modelling on the release text of impactful releases. In this table, we consider only those apps for which release text is available. For Google, of the 754 target releases (those with sufficient evidence for causal impact analysis), 641 have release text available. For Windows, 546 of the 793 target releases have available release text. The remainder of the table consists of four overall columns, giving the metric for which a release is found to be impactful (leftmost column), followed by the most prevalent terms (for TF.IDF) and topics (for topic modelling), followed by a subdivision of these prevalent terms into those whose impacts are positive and negative from the developers' perspective.

Table 5 reveals that terms and topics themed around bug fixes occur frequently in the Google dataset, while in the Windows dataset, the topics appear to be more closely associated with features (message chat free, new search feature).

The Windows app store is comparatively more recent than the Google app store, and it is tempting to speculate that, at this comparatively immature stage, perhaps users are more concerned with new features than bug fixes. Further research would be required to investigate this possibility. Nevertheless, these observations motivate our analysis in RQ4.2 for the tuples (bug, fix) and (new, feature).

RQ4.2: How often do top terms and topics occur in each set of releases? Table 6 shows the number of occurrences, within impactful and non-impactful releases, of the tuples (bug, fix) and (new, feature) highlighted by information retrieval in RQ4.1. We can see from the results in Table 6 that the terms (bug, fix) are more common in the non-impactful releases in both stores. However, it is more striking that for the metrics (R)ating and (D)ownload, the terms occur more often in the releases that negatively impact rating, and negatively impact popularity (having a positive impact on (D)ownloads column). We conclude, therefore, that release text mentioning bug fixes occurs more frequently in releases that negatively impact metrics such as rating and download rank.

The results show that there are proportionally more mentions of (new, feature) in releases that positively impact (R)ating and popularity for both Google and Windows. There were (proportionally) more impactful releases in the Google store that mentioned (new, feature), but fewer for Windows.

Our results lead to the conclusions that releases are more likely to positively impact rating or popularity if they claim to introduce new features, and more likely to negatively impact rating or popularity if they claim to fix bugs. While the former finding is to be expected, it seems a little unfair on developers that bug fix claims might reduce performance. Future work might further investigate this effect to see whether bug fix claims are unsubstantiated, thereby providing a potential explanation.

6 Threats to Validity

In this section we discuss threats to the construct, conclusion and external validity of our findings.

Construct Validity: The gap between data analysis and causality is large, forcing any user to make very strong assumptions if they hope to effectively imply causality. Causal analysis can hope to reduce this gap, but no such analysis could ever hope to fully close the gap; there will always be unknown factors which may nevertheless have affected the data and the scrutiny. In the specific case of app stores, there will always be potential external influences for which no data is available to capture them.

We have shown how causal impact analysis can be a useful way to identify releases that have some form of impact. This may be useful to the developers in its own right, since they might choose to further manually analyse those releases for which evidence for impact is strong. Nevertheless, the developer would undoubtedly be more interested to know the *precise causes* of an impact. Unfortunately, this is simply overoptimistic; because there is no necessary link between correlation and causation, the best we can hope to do is provide probabilistic evidence that ranks potential causes as we have done in RQ3 and RQ4. Great care is required when interpreting these probabilistic findings, particularly when they are applied to the analysis of a specific app release for which one or more external factors may have had some particular effect.

Conclusion Validity: Our conclusion validity could be affected by the qualitative human assessment of 'top terms' and topics for sets of releases in RQ4.1. We mitigate against this threat to validity by asking a quantitative question of the number of times (bug, fix) and (new, feature) occur in each set of releases in RQ4.2.

External Validity: Our dataset is subject to the App Sampling Problem [35]. Performance data is available only for the most popular apps in each of the app stores studied. We restrict our claims about findings to those that apply specifically to the most consistently popular apps over the 52 week period studied, and thereby do not suffer from the App Sampling Problem in our findings. However, any attempt to extend and generalise the findings to other apps, would be vulnerable to the App Sampling Problem, and so great care is required due to this potential threat to validity.

Table 5: RQ4.1: Top release text terms: TF.IDF terms on the left and Topic Modelling topics on the right.

Type	**Overall**		**Apps**	307		
Non-impactful	new fix bug	fix bug fixed	**Target releases**	754		
Impactful	fix bug new	fix bug fixed	**Release text**	641		
Metric	**All**		**+ve**		**-ve**	
R	fix bug new	sky device channel	sky fix bug	sky device channel	fix bug new	app account use
D	fix bug new	new game experience	new game security	character new power	fix bug new	new game experience
N	new fix bug	sky device channel	new fix bug	sky device channel		
NW	new fix improvement	photo filter add	new performance improvement	app music live	fix bug improvement	photo filter add
R ∩ D	fix bug sky	app account use	sky fruit carp	sky device channel	bug fix pay	app account use
R ∩ D ∩ NW	various klingon improvement	android song facebook	security traveler dim	app purchase flight	song piano smule	android song facebook

Google

Type	**Overall**		**Apps**	726		
Non-impactful	video photo new	message chat free	**Target releases**	793		
Impactful	video app new	file medium server	**Release text**	546		
Metric	**All**		**+ve**		**-ve**	
R	video app phone	added app sound	video music phone	message chat free	app video photo	added app sound
D	video app phone	app live new	video new app	app live new	video phone app	message chat free
N	video app phone	file medium server	video app phone	file medium server		
NW	app video dating	app apps music	dating video app	added app sound	weather shazam app	app apps music
R ∩ D	video app phone	message chat free	video music task	video fix youtube	channel app phone	app also account
R ∩ D ∩ NW	hike dating event	message chat free	dating pof free	message chat free	hike learn learning	message chat free

Windows

Table 6: RQ4.2: Occurrences of (bug, fix) and (new, feature) in release text.

Type	**Overall**	**Apps**	307
Non-impactful	145 / 379 (38.3%)	**Target releases**	754
Impactful	87 / 262 (33.2%)	**Release text**	641
Metric	**Total**	**+ve**	**-ve**
R	41 / 133 (30.8%)	16 / 56 (28.6%)	25 / 77 (32.5%)
D	42 / 110 (38.2%)	10 / 36 (27.8%)	32 / 74 (43.2%)
N	17 / 46 (37.0%)	17 / 46 (37.0%)	0 / 0
NW	22 / 77 (28.6%)	13 / 48 (27.1%)	9 / 29 (31.0%)
R ∩ D	12 / 29 (41.4%)	2 / 7 (28.6%)	4 / 12 (33.3%)
R ∩ D ∩ NW	3 / 7 (42.9%)	1 / 2 (50.0%)	0 / 1

Google (bug, fix)

Type	**Overall**	**Apps**	726
Non-impactful	118 / 248 (47.6%)	**Target releases**	793
Impactful	130 / 298 (43.6%)	**Release text**	546
Metric	**Total**	**+ve**	**-ve**
R	62 / 152 (40.8%)	22 / 58 (37.9%)	40 / 94 (42.6%)
D	50 / 137 (36.5%)	22 / 71 (31.0%)	28 / 66 (42.4%)
N	77 / 183 (42.1%)	77 / 183 (42.1%)	0 / 0
NW	14 / 33 (42.4%)	8 / 18 (44.4%)	6 / 15 (40.0%)
R ∩ D	18 / 57 (31.6%)	2 / 9 (22.2%)	9 / 18 (50.0%)
R ∩ D ∩ NW	1 / 8 (12.5%)	0 / 1	1 / 3 (33.3%)

Windows (bug, fix)

Type	**Overall**	**Apps**	307
Non-impactful	92 / 379 (24.3%)	**Target releases**	754
Impactful	72 / 262 (27.5%)	**Release text**	641
Metric	**Total**	**+ve**	**-ve**
R	36 / 133 (27.1%)	17 / 56 (30.4%)	19 / 77 (24.7%)
D	34 / 110 (30.9%)	13 / 36 (36.1%)	21 / 74 (28.4%)
N	12 / 46 (26.1%)	12 / 46 (26.1%)	0 / 0
NW	21 / 77 (27.3%)	14 / 48 (29.2%)	7 / 29 (24.1%)
R ∩ D	10 / 29 (34.5%)	4 / 7 (57.1%)	2 / 12 (16.7%)
R ∩ D ∩ NW	3 / 7 (42.9%)	1 / 2 (50.0%)	0 / 1

Google (new, feature)

Type	**Overall**	**Apps**	726
Non-impactful	133 / 248 (53.6%)	**Target releases**	793
Impactful	145 / 298 (48.7%)	**Release text**	546
Metric	**Total**	**+ve**	**-ve**
R	69 / 152 (45.4%)	32 / 58 (55.2%)	37 / 94 (39.4%)
D	68 / 137 (49.6%)	39 / 71 (54.9%)	29 / 66 (43.9%)
N	95 / 183 (51.9%)	95 / 183 (51.9%)	0 / 0
NW	16 / 33 (48.5%)	8 / 18 (44.4%)	8 / 15 (53.3%)
R ∩ D	27 / 57 (47.4%)	6 / 9 (66.7%)	7 / 18 (38.9%)
R ∩ D ∩ NW	4 / 8 (50.0%)	1 / 1 (100.0%)	2 / 3 (66.7%)

Windows (new, feature)

Naturally, care is also required when extending our findings to other app stores. Indeed, our study on the Google and Windows stores shows differences between the two, as higher priced releases in Google are more likely to be impactful, whilst the same is true of lower priced releases in Windows. There are many common findings, but this difference highlights the fact that our results may not apply to other app stores. Nevertheless, the methods we used to analyse causal impacts can be applied to other app stores.

7 Related Work

App Store Repository Mining [20, 37, 44, 49, 52] is a form of software repository mining. Recent papers in the field have compared app stores [47], and studied code reuse [43]. Many studies have also extracted API information for malware detection [1, 15], detecting energy greedy APIs [30, 31], and assessing how API usage affects user ratings [2, 42] and code quality [48]. There has been much recent work in app store review analysis that mines user reviews from app stores for requirement prioritisation and elicitation [14, 24, 40], summarisation [8, 13, 16, 23, 32, 38] and even for test device prioritisation [26]. Other work incorporated sentiment analysis in order to identify complaint types [25, 27] and differences in comments lodged in different stores [19]. Recent work has also studied how responding to user feedback can increase the rating [39].

Martin et al. [35] introduced the concept of the App Sampling Problem, a threat to validity for app store analysis due to the inherent biases associated with partial information. This threat can be ameliorated using inferential statistical techniques and/or by carefully defining the scope of claims about apps. In this paper, we avoided the App Sampling Problem by limiting our claims to the most consistently popular apps in each of the two app stores.

There has been considerable previous work that studies app releases: In 2011 Henze and Boll [21] analysed release times and user activity in the Apple App Store, and concluded that Sunday evening is the best time for deploying games. In 2013 Datta and Kajanan [12] studied review counts from the Apple App Store, and found that apps receive more reviews after deploying updates on Thursday or late in the week. In 2015 Gui et al. found from 21 apps from Google Play with frequent releases, that 23% of their releases contained ad-related changes [18]. Comino et al. [9] studied the top 1000 apps in Apple App Store and Google Play, finding that app releases are more likely when the app is performing badly, and that releases can boost downloads. Very recently, McIlroy et al. [36] studied update frequencies in the Google Play store after mining data about 10,713 mobile apps. They found that only 1% of apps received more than one update per week, and only 14% of the studied apps were updated in a two-week period. These findings support our weekly data collection schedule, as very few releases can be 'missed' by collecting data weekly; additionally the target releases we use (defined in Section 4.3), mandate that very frequently updated apps are excluded due to lack of sufficient prior and posterior time series data. McIlroy et al. [36] also found that rating was not affected by update frequency, however the findings by Guerrouj et al. [17] indicate that high code churn in releases correlates with lower ratings.

All of these previous findings on app releases tantalisingly point to the possibility that certain releases may have higher impact than others. Taken together, this previous work is one of the motivations for the present paper. However, no previous paper has specifically addressed the question of how we identify, in general, those releases that are impactful, nor has any previous work attempted to identify the characteristics of highly impactful app releases: the primary technical and scientific contributions of the present paper.

Causal inference is not a new concept, and whilst used primarily in economic forecasting, it has been discussed in empirical science papers [29]. As such, we are not the first software engineers to use the concept: see for example the work using the Granger causality test by Couto et al. [10, 11] and Zheng et al. [53] as a means of software defect prediction, and the work by Ceccarelli et al. [7] on identifying software artifacts affected by a change. We are, however, the first authors to apply causal impact analysis to app store analysis.

The technique we use for causal impact analysis was introduced by Brodersen et al. [6]. They exploited the Bayesian structural time series model for causal inference, motivated by the need to measure the success of online advertising campaigns. The method differs from the Granger test, which measures the degree to which a vector can be used to predict another. Instead, Brodersen's approach (which we use in the present paper) creates and compares a counterfactual prediction with the observed, in order to establish whether an event could have significantly altered the path of the vector. For more information about recent advances in causal inference, we refer the reader to the review by Maathuis and Nandy [33].

8 Conclusions

Our analysis of the Google Play and Windows Phone app stores, over a period of 52 weeks from July 2014 to July 2015 has identified individual releases that caused a significant impact in rating, download rank, number of ratings or number of ratings per week. Overall, 40% of releases had an impact in Google and 55% had an impact in Windows. Our research has found that overall release frequency is not correlated with subsequent app performance, but that there is evidence that release text content plays a role in whether a release is impactful and the type of impact it has. Releases with text mentioning new features instead of bug fixes are more likely to be impactful and to positively impact rating.

References

[1] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden. Mining apps for abnormal usage of sensitive data. In *37th*

International Conference on Software Engineering (ICSE'15), 2015.

[2] G. Bavota, M. Linares-Vasquez, C. E. Bernal-Cardenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk. The impact of api change-and fault-proneness on the user ratings of Android apps. *IEEE Transactions on Software Engineering*, 41(4):384–407, 2015.

[3] Y. Bejamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal statistical Society (Series B)*, 57(1):289–300, 1995.

[4] D. M. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[5] K. H. Brodersen. CausalImpact. `https://google.github.io/CausalImpact/CausalImpact.html`. Retrieved 28th May 2015.

[6] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott. Inferring causal impact using bayesian structural time-series models. *Annals of Applied Statistics*, 9:247–274, 2015.

[7] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta. An eclectic approach for change impact analysis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 163–166. ACM, 2010.

[8] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, pages 767–778, 2014.

[9] S. Comino, F. M. Manenti, and F. Mariuzzo. Updates Management in Mobile Applications. iTunes vs Google Play. *Centre for Competition Policy (CCP), University of East Anglia*, 2015.

[10] C. Couto, P. Pires, M. T. Valente, R. S. Bigonha, and N. Anquetil. Predicting software defects with causality tests. *Journal of Systems and Software*, 93:24–41, 2014.

[11] C. Couto, C. Silva, M. T. Valente, R. Bigonha, and N. Anquetil. Uncovering causal relationships between software metrics and bugs. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR'12)*, pages 223–232, 2012.

[12] D. Datta and S. Kajanan. Do app launch times impact their subsequent commercial success? an analytical approach. In *International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, pages 205–210. IEEE, 2013.

[13] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 1276–1284, 2013.

[14] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*, pages 582–591, 2013.

[15] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 2014 International Conference on Software Engineering (ICSE'14)*, pages 292–302, 2014.

[16] X. Gu and S. Kim. What parts of your apps are loved by users? In *In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, 2015.

[17] L. Guerrouj, S. Azad, and P. C. Rigby. The influence of App churn on App success and StackOverflow discussions. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, pages 321–330, 2015.

[18] J. Gui, S. Mcilroy, M. Nagappan, and W. G. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, 2015.

[19] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014.

[20] M. Harman, Y. Jia, and Y. Zhang. App Store Mining and Analysis: MSR for App Stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*, pages 108–111, 2012.

[21] N. Henze and S. Boll. Release your app on sunday eve: Finding the best time to deploy apps. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI'11)*, pages 581–586, 2011.

[22] P. W. Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81(396):pp. 945–960, 1986.

[23] L. Hoon, R. Vasa, J.-G. Schneider, and J. Grundy. An analysis of the mobile app review landscape: trends and implications. Technical report, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2013.

[24] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*, pages 41–44, 2013.

[25] H. Khalid. On identifying user complaints of ios apps. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, 2013.

[26] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan. Prioritizing the devices to test your app on: A case study of Android game apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, pages 610–620, 2014.

[27] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[28] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams. Do faster releases improve software quality? an empirical case study of Mozilla Firefox. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*, pages 179–188, 2012.

[29] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 273–281, Washington, DC, USA, 2004. IEEE Computer Society.

[30] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: A threat to the success of Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*, pages 477–487, 2013.

[31] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Mining energy-greedy api usage patterns in Android apps: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 2–11. ACM, 2014.

[32] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *Proceedings of the Requirements Engineering Conference, 23rd IEEE International (RE'15)*, 2015. To appear.

[33] M. H. Maathuis and P. Nandy. A review of some recent advances in causal inference. *arXiv preprint arXiv:1506.07669*, 2015.

[34] C. D. Manning, P. Raghavan, and H. Schütze. Scoring, term weighting, and the vector space model. In *Introduction to Information Retrieval*, pages 100–123. Cambridge University Press, 2008.

[35] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories (MSR'15)*, pages 123–133, 2015.

[36] S. McIlroy, N. Ali, and A. E. Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, pages 1–25, 2015.

[37] R. Minelli and M. Lanza. Samoa – a visual software analytics platform for mobile applications. In *Proceedings of 29th International Conference on Software Maintenance (ICSM'13)*, 2013.

[38] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE'13)*, 2013.

[39] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps. In *Proceedings of 31st IEEE International Conference on Software Maintenance and Evolution (ICSME'15)*, 2015. To appear.

[40] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall. How can I improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of 31st IEEE International Conference on Software Maintenance and Evolution (ICSME'15)*, 2015. To appear.

[41] K. Pearson. Notes on regression and inheritance in the case of two parents. *Proc. of the Royal Society of London*, 58:240–242, June 1895.

[42] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. Impact of ad libraries on ratings of Android mobile apps. *IEEE Software*, 31(6):86–92, Nov. 2014.

[43] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan. Understanding reuse in the Android market. In *ICPC*, pages 113–122, 2012.

[44] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain and die in app stores. In *Proceedings of the Requirements Engineering Conference, 23rd IEEE International (RE'15)*. IEEE, 2015. To appear.

[45] S. L. Scott and H. R. Varian. Predicting the present with bayesian structural time series. *International Journal of Mathematical Modelling and Numerical Optimisation*, 5(1-2):4–23, 2014.

[46] C. E. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[47] M. D. Syer, B. Adams, Y. Zou, and A. E. Hassan. Exploring the development of micro-apps: A case study on the blackberry and Android platforms. In *Proceedings of the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation (SCAM'11)*, pages 55–64, 2011.

[48] M. D. Syer, M. Nagappan, B. Adams, and A. E. Hassan. Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal (SQJ)*, 2014. To appear.

[49] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source Android apps. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'13)*, pages 283–297, 2013.

[50] A. Vargha and H. D. Delaney. A critique and improvement of the "cl" common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):pp. 101–132, 2000.

[51] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[52] Y. Yang, J. Stella Sun, and M. W. Berry. APPIC: Finding The Hidden Scene Behind Description Files for Android Apps. Technical report, Dept. of Electrical Engineering and Computer Science University of Tennessee, 2014.

[53] P. Zheng, Y. Zhou, M. R. Lyu, and Y. Qi. Granger causality-aware prediction and diagnosis of software degradation. In *IEEE International Conference on Services Computing (SCC'14)*, pages 528–535, 2014.