On the Relationship between User Churn and Software Issues

Omar El Zarif
oelzarif@cs.queensu.ca
Queen's University
School Of Computing
Kingston, Ontario, Canada

Daniel Alencar Da Costa
danielcalencar@otago.ac.nz
University of Otago
Department of Information Science
Dunedin, Otago, New Zealand

Safwat Hassan
shassan@cs.queensu.ca
Queen's University
School Of Computing
Kingston, Ontario, Canada

Ying Zou
ying.zou@queensu.ca
Queen's University
Department of Electrical and Computer Engineering
Kingston, Ontario, Canada

ABSTRACT

The satisfaction of users is only part of the success of a software product, since a strong competition can easily detract users from a software product/service. *User churn* is the jargon used to denote when a user changes from a product/service to the one offered by the competition. In this study, we empirically investigate the relationship between the issues that are present in a software product and user churn. For this purpose, we investigate a new dataset provided by the alternativeto.net platform. Alternativeto.net has a unique feature that allows users to recommend alternatives for a specific software product, which signals the intention to switch from one software product to another. Through our empirical study, we observe that (i) the intention to change software is tightly associated to the issues that are present in these software; (ii) we can predict the rate of potential churn using machine learning models; (iii) the longer the issue takes to be fixed, the higher the chances of user churn; and (iv) issues within more general software modules are more likely to be associated with user churn. Our study can provide more insights on the prioritization of issues that need to be fixed to proactively minimize the chances of user churn.

KEYWORDS

software issues, users churn, software alternatives, deep learning

ACM Reference Format:
Omar El Zarif, Daniel Alencar Da Costa, Safwat Hassan, and Ying Zou. 2020. On the Relationship between User Churn and Software Issues. In *17th International Conference on Mining Software Repositories (MSR '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3379597.3387456

MSR'20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7517-7/20/05…$15.00
https://doi.org/10.1145/3379597.3387456

1 INTRODUCTION

User satisfaction is essential for the success of software projects [22, 23, 60]. In fact, even when the budget and the schedule of a project are in control, user dissatisfaction may still lead the project to failure [43, 76]. Users tend to discuss in public (e.g., public forums) their preferences and disappointments regarding software, in which they often mention what made them love or hate a software product (e.g., by providing ratings and detailed comments). Such data about user satisfaction has been continuously used by researchers to study the most important factors to explain user satisfaction. For example, Panichella et al. [52] identified useful user reviews of mobile apps, so that developers can improve their apps accordingly (e.g., by addressing feature requests within such reviews). Other research works have extracted user feedback [31], and studied the planning process of future releases based on user reviews [72].

Nevertheless, the success of software projects is not only defined by the relationship between the software product and its users, but also by the strengths and weaknesses of competitors. For instance, studies have shown that a poor user experience may create a point of no return in which the user is led to the competition or searches for alternative solutions [62]. In addition, a poor user experience may create a bad reputation for the software product, which impairs the adherence of new users [11] (which would likely adhere to the competitors). Therefore, an important area of study is to unveil the underlying reasons for losing users to competitors. *User churn* is the jargon used to denote when a user decides to change from a product/service to those offered by the competition.

User churn has been studied extensively in areas other than software engineering, such as mobile operators and telecommunication networks [20, 53, 73]. In fact, Wei et al. [73] observed that the cost of acquiring new users is more expensive than retaining users. Several services investigate their own user base characteristics to predict user churn. There has also been research targeting Yahoo Answers [24], StumbleUpon (a web content recommendation system) [21], Top Eleven - Be A Football Manager (an online mobile game) [41], Pengyou (a Chinese social network) [37], using prediction models for predicting user churn.

Although prior research has studied the concerns of users regarding software products (including mobile apps), there has been a lack of empirical research to investigate user churn in the context of software products. Filling this gap is important to help open

source and corporate software organizations to retain their users and improve their probability of success (not to mention the overall satisfaction of users).

In this paper, we use data obtained from the *alternativeto.net*[1] website, which has a unique feature that allows users to recommend alternatives for a specific software product. The recommendation of alternatives can signal the intention to switch from one software product to another (we provide an example of this scenario in Section 2). For the sake of simplicity, we refer to the recommendation for an alternative software product as simply *potential user churn*.

By using the *alternativeto.net* dataset, we formulate an empirical study to investigate the (i) Web Browsers, (ii) IDEs and (iii) Web Servers domains on the *alternativeto.net* website. We first extract 3,556 reviews and 10,081 comments to better understand the overall concerns of users regarding the software products in the studied domains. Next, we extract 12,549 recommendations for alternatives by users (i.e., *potential user churn*) from the Firefox, Eclipse, and Apache projects. We specifically choose these projects for the deeper analyses because these projects are established in the *Web Browser*, *IDE* and *Web Server* domains, respectively.

To investigate the underlying reasons for the potential user churn of software products, we pose four research questions that guides our study:

- *RQ0: What are the most significant users' concerns for Browsers, IDEs and Web Servers?* Understanding users' concerns will pave the way to grasp the reasons behind potential user churn. When discussing the pros and cons of a software product and competitors, users voice varied concerns (e.g., security & privacy issues or memory issues).
- *RQ1: Can we find a correlation between potential user churn and issue reports?* Given that the issues that are present in the studied projects are the predominant topic in users' concerns, we study whether there exists a correlation between the issue reports of a given software project (i.e., Firefox, Eclipse, and Apache) and the potential user churn.
- *RQ2: Can we use machine learning models to predict the rate of potential user churn?* After correlating issue reports with potential user churn, we study whether machine learning can be used to predict the rate of potential user churn (expressed by high or low user churn). To build our machine learning models, we use latent features from the issue reports of our studied project.
- *RQ3: What are the most important factors related to potential user churn?* given that machine learning models can be used to predict the rate of potential user churn, we can then extract the most important features that are associated to the potential user churn. The *reoccurring bugs*, the *time alive* factor, and the *issues across multiple platforms* are common factors that are highly associated with the potential user churn.

The paper is organized as the following, we showcase a motivating example in Section 2. We describe our data collection process in Section 3. We delve into our approach and show our obtained results in Section 4. We discuss the threats to validity in Section 5. In Section 6, we present the related work. Finally, we conclude the study in Section 7.

[1]https:\alternativeto.net

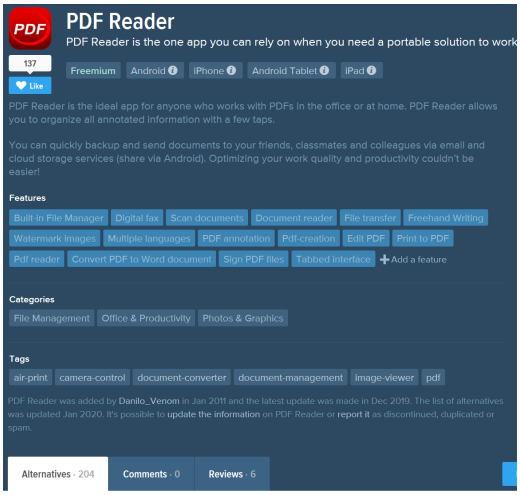

Figure 1: Example of a pdf reader on the *alternativeto.net* website.

2 MOTIVATING EXAMPLE

The goal of the *alternativeto.net* website is to help users to find software alternatives that can better address the users' necessities. For instance, let us consider that a user needs a better *.pdf reader* (e.g., the current reader freezes occasionally). The first challenge occurs because the user is not aware of all the available software alternatives. In addition, choosing an alternative for a software is not always simple, since users may be already familiar with a set of features (from the software in use), which they would not like to compromise. Considering the .pdf reader example, while the user wishes a freezing-free alternative, the user may only feel comfortable to change if the alternative provides the same level of commenting capabilities (as compared to the reader in use).

With such challenges in mind, the *alternativeto.net* website was designed to help users choose the best software alternative for their needs. Figure 1 shows an example of an initial page of a software product on *alternativeto.net*. This initial page provides a description of the software, *features*, *categories*, and *tags*. The *features*, *categories* and *tags* are used to organize the software alternatives on *alternativeto.net* and can be used to search for software alternatives. Interesting to note, is the *"Alternatives"* tab depicted in Figure 1, which shows all the available alternatives for a software product (as deemed by the community).

Alternativeto.net allows users to provide reviews for software alternatives along with ratings (similarly to Google Play, which allows reviews to be provided for mobile apps). However, what sets *alternativeto.net* apart from other platforms is that it allows users to voice their opinions by placing a software product in perspective to its competitors. For example, Figure 2 shows the opinions voiced by users as to whether (and why) Chrome is a good (or bad) alternative to Firefox.[2]

As shown in Figure 2, users do not only comment on the product being evaluated, but also put the product in perspective to its

[2]https://alternativeto.net/software/firefox/

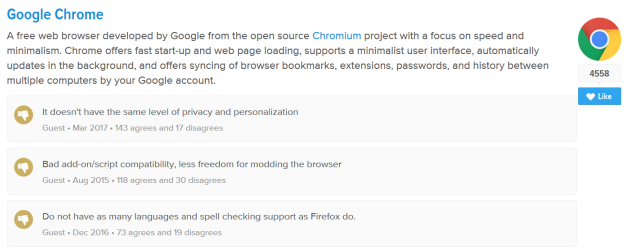

Figure 2: https://alternativeto.net provides *in-perspective* comments where users specify why a certain software alternative (e.g., Firefox) can be better than the current software (e.g., Chrome)

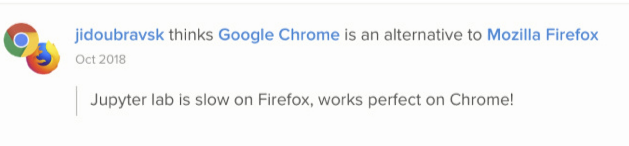

Figure 3: Recording the recommendation for an alternative for Firefox

competitors. For example, a user mentions *"[Chrome] does not have as many languages and spell checking support as Firefox does.*[3]"

Because the opinions are voiced with respect to competitors (i.e., the in-perspective comments), *alternativeto.net* provides a unique opportunity for empirical studies. Particularly in our study, the in-perspective comments allow us to investigate the most recurrent concerns within the competition scene of a software domain (we perform this investigation in RQ0).

Another interesting and unique characteristic in the *aternativeto.net* data is the register of users that recommended alternatives for a given software product (i.e., the potential user churn). For example, Figure 3 shows how *alternativeto.net* records when a user indicates that Chrome can be a good alternative to the Firefox web browser (along with the reason why). This data is important because it can be used to proactively identify potential user churn in software products (e.g., by using machine learning). Based on this information, we build machine learning models and report on their results in RQ2 and RQ3.

Finally, the potential user churn provided by *alternativeto.net* allow us to investigate whether it is related to prominent issues that may be present in a software product. For example, while a user has expressed the reason for a possible user churn in Figure 3 (i.e., Jupyter is slow on Firefox), the development team of Firefox is also working on an issue report that expresses the same issue (see Figure 4). In this paper, we hypothesize that *there may exist strong links between potential user churn and the issue reports that are submitted to software products* (we perform this investigation in RQ1).

[3]To provide an opinion as to whether a software product is a good or a bad alternative to another product, users must prove they are not robots. Only afterwards, a text box containing the submission form will be provided.

Figure 4: Issue report expressing the problem voiced by the user in Figure 3

We capitalize on these unique characteristics of the *alternativeto.net* data to perform our empirical study on the relationship between potential user churn and software issues.

3 DATA COLLECTION

Alternativeto.net organizes the software alternatives through several means, such as *categories*, *features*, or *tags*. *Categories* are the broadest way to categorize software alternatives and *alternativeto.net* has over 25 categories of software alternatives at the time of this study. *Tags* and *features* are more specific ways to categorize software alternatives (e.g., the "IDE" tag or the "responsive design" feature). Another characteristic of *tags* and *features* is that they can be added by users. However, these new *tags* and *features* must be verified by the *alternativeto.net* community before being added to the website. For the purposes of our study, we use *tags* to filter our data for software alternatives because it is more precise than using *categories*. For example, we can easily filter for the software alternatives within the Web Browser domain using the "web browser" tag. On the other hand, it would be hard to filter our data based on the *features*. For example, filtering by "responsive design" may retrieve software from very distinct domains.

For the sake of data robustness, it is tempting to analyze all the software alternatives from all the software domains that are available on the *alternativeto.net* website. However, since one of our main objectives is to study the relationship between potential user churn and the issues that are present in the software products, we need to restrict our analyses to the domains of certain software. For instance, it would be virtually impossible to collect and analyze all the issue reports from every software alternative listed on the *alternativeto.net* website. Every different software alternative may use a different Issue Tracking System, which would require different ways for collecting data. Therefore, we analyze the software alternatives within the domains of three well established open source projects: Eclipse,[4] Firefox,[5] and the Apache server.[6] Our choice for these three projects was based on the extensive prior empirical software engineering research that has been conducted using these projects over the years [2, 7, 18, 19, 29, 36, 42, 54].

To collect the data for our study, we followed the process depicted in Figure 5. Given our chosen studied projects, we collect all the software alternatives tagged with the "Browser", "IDE", and "Web Server" tags, which are the respective domains of the Eclipse, Firefox, and Apache projects. In total, we collect 290 browser alternatives, 296 IDE alternatives, and 134 web-server alternatives. Once the software alternatives are fetched, we collect the *in-perspective*

[4]https://www.eclipse.org/
[5]https://www.mozilla.org/en-US/firefox/new/
[6]https://httpd.apache.org/

Table 1: A summary of our collected data.

Domain	Alternatives	Reviews	Comments	Potential churn	Issue reports
IDEs	296	1,088	3,036	4,319 (Eclipse)	22,758 (Eclipse)
Browsers	290	1,754	5,112	6,421 (Firefox)	40,749 (Firefox)
Web Servers	124	714	1,933	1,809 (Apache)	20,965 (Apache)

comments (explained in Section 2) and the reviews for all the software alternatives. We also collect the *potential user churn* (explained in Section 2) for the Eclipse, Firefox, and Apache projects. Finally, in addition to the data collected from *alternativeto.net*, we also collect the issue reports from the respective Issue Tracking Systems of our studied projects. We use the *potential user churn* data combined with the *issue reports* to perform our investigations in RQ1-RQ3. Table 1 summarizes the data collected for our study (i.e., the data type and the amount of collected data).

4 RESEARCH QUESTIONS & RESULTS

In this section, we present our research questions (RQs) along with their results. For each RQ, we present its motivation and explain the approach that is used to conduct the RQ.

RQ0: What are the most significant users' concerns for Browsers, IDEs and Web Servers?

Motivation: The motivation behind RQ0 is to understand the main concerns within the competition scene of a specific software domain. This investigation is important to highlight what are the main strengths and weaknesses within a particular domain (e.g., Browsers). This knowledge can be useful because, for example, an organization can study the main weaknesses within a domain and better assess whether its products/services fall within the same flaws.

Approach: To study the most significant concerns related to Browsers, IDEs and Web Servers, we use the *in-perspect comments* and *reviews* data that were explained in Section 3. To analyze such data, we adopt the Latent Dirichlet Allocation (LDA) to find the main topics that are present in the in-perspective comments and reviews data. We use the LDA implementation provided by the MALLET framework [40]. LDA is a probabilistic generative method that assumes a Dirichlet distribution of latent topics. LDA is particular useful for us, since we are interested in finding the main topics that are present in the *in-perspective comments* and *reviews* data. However, The LDA model requires the data to be pre-processed in a certain format for optimal results. Therefore, we follow the standard text pre-processing steps [48]:

Fixing Typographical Errors. Typographical errors can be due to the usage of internet slang in a written text or simply due to common language mistakes. The most recurrent of errors that we find in our data is the repetition of characters. For example, users tend to use words, such as "beeeest" to emphasize their feelings. We use the `Pattern For Python` [63] package to fix such typographical errors.

Removing Stop Words: Examples of stop words are "of", "by" and "the". These are words that are recurrently used in written text, but do not possess meaning by themselves. We use the package `nltk` [8] to remove such stop words. This package contains a well-known corpus of stop words and is widely used for text filtering.

Stemming and Lemmatization. For better results, words should be repeated as frequently as they can, since LDA relies heavily on the occurrences of words in different sentences. Hence, the usage of stemming and lemmatization is essential. These methods transform inflectional forms of a word to its basic form. For example, the words "fixed", "fixing", and "fixes" are transformed to the basic form "fix." Stemming works by removing the suffix of the word (e.g., the word "cats" becomes "cat" by removing the "s"), while lemmatization retrieves the basic form of a word from a dictionary known as lemma (by using morphological analysis). We use both stemming and lemmatization to ensure that we obtain the most unique words possible.

Forming Bigrams and Trigrams. Forming bigrams and trigrams is essential to group the words which frequently appear together. Some words do not inflict any significant meaning for the LDA model on their own. However, when grouped together, such words may indicate a specific topic. One example of these words is "customer support," which has a precise meaning (as opposed to only "customer" or "support", which can hold diverse meanings).

To choose the optimal number of topics in our LDA model, we use the coherence score metric [56]. This metric was found to be the best in alliance with human perception [49, 65]. However, even when relying on the coherence score (which naturally limits redundancy in the produced topics), there is still the possibility of duplicate topics being generated. Therefore, we perform a manual analysis of the produced topics to eliminate duplicates. Our LDA model produces topics in the form of keywords and their percentage-of-significance with respect to the topic. For example, $(0.097 * \text{"}source\text{"} + 0.085 * \text{"}privacy\text{"} + 0.053 * \text{"}track\text{"} + 0.050 * \text{"}profit\text{"} + 0.038 * \text{"}quantum\text{"} + 0.038 * \text{"}collect\text{"} + 0.035 * \text{"}protect\text{"} + 0.032 * \text{"}archive\text{"} + 0.029 * \text{"}money\text{"} + 0.026 * \text{"}performance\text{"})$ are a set of keywords that indicate *Privacy* as high level topic. But some other distribution of keywords could refer to the same topic. Two of the authors conduct together the manual analyses of the topics. The final set of topics is only produced when full consensus is reached between authors (i.e., the authors went over and discussed all the topics). Because both authors have analyzed together all topics and agreements were reached on all the topics in the end, we did not compute inter-rater agreement.

Once the topics are defined, we extract 100 examples (aiming for a 95% confidence level with a 10% margin error) from our data for each studied domain (i.e., IDE, Browser, and Web Server), totalling 300 examples. For each example, two of the authors assign their topic. Again, two authors analyze and discuss each example so that consensus is reached regarding the assigned topic. Afterwards, we compute the topic distribution for each domain, which can represent the most recurrent concerns for that domain.

Findings: Tables 2 and 3 show the topics obtained by our LDA model. We also show some of the examples for each topic in the table (all the examples and analyses are available in our replication package, which is hosted on https://doi.org/10.5281/zenodo.3610584.

Our obtained topics can be categorized into (i) general topics, which are present in the three domains; and (ii) domain-specific topics, which might still be present in other domains but are more

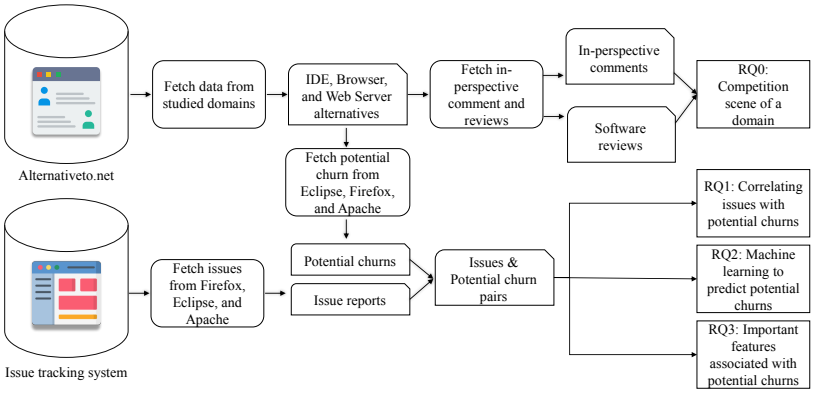

Figure 5: An overview of our data collection process.

dominant in a specific domain. Table 4 shows the distribution of topics per domain based on our manual analyses. We observe that *release and updates*, *memory issues*, *bugs/crashes*, and *cross-platform availability* can be considered as general topics (i.e., they are important in all the three analyzed domains). Indeed, these topics are intuitively present when handling any kind of software.

On the other hand, *Documentation* and *testing* were present for web servers and IDEs, signalling that a robust documentation (which is often overlooked [55]) and with good support for testing tools may definitely attract/retain more users when competing in the IDE and Web Sever Market. For example, an organization may direct their attention to these aspects when studying their competition.

The *bookmarks*, *privacy*, and *security* were exclusively present in the Browsers domain, signalling that these are essential features to retain or attract users. *Simplistic design*, *debug options*, *code refactoring and auto-completion* are topics that target IDEs exclusively.

Finally, complaints about *release and updates*, *memory issues* and *bugs/crashes* can be considered as the main topics that drive users to churn. These topics were present in the three domains with the highest percentages among each of the domains. The percentages combined exceed 50% in each of the domains (see Table 4).

Summary: Our observations suggest that the users' concerns are tightly related to issues (e.g., bugs or crashes) that are present in the respective software products.

RQ1: Can we find a correlation between potential user churn and issue reports?

Motivation: In RQ0, we observe that software issues (e.g., bugs & crashes and memory issues) are common concerns that may motivate users to churn. With such an observation, we hypothesize in RQ1 that the *potential users'churn* observed on *alternativeto.net* may be correlated with the issue reports which developers address in their respective projects. This analysis is important because if such a correlation exists, the *potential user churn* information can be possibly used to prioritize issue reports (for example, the more associated with potential user churn the more important the issue report).

Approach: In our work, we approach the recommendations for alternatives (i.e., potential users'churn) and the issue reports as *events occurring over time* and interpret them as two time-series. Such an interpretation is handy for two reasons: (i) we can study the correlation between two trends (potential users'churn and issue reports); and (ii) we can study whether the specific peaks in issue reports can be associated with peaks in *potential users'churn*.

In our study, we group the two time series in a weekly basis, i.e., all the potential user churn that occurred within a week are grouped together. We use the weekly grouping because grouping issue reports in a daily-basis is too narrow to form a pattern, while grouping them in a monthly-basis can over-generalize the patterns in issues. The time-series of potential users'churn and the time-series of issues showcase the events (grouped in a weekly-basis) from March 2015 (the earliest date on *alternativeto.net*) to July 2019, forming 200 weeks grouping.

The first step to correlated our two time-series is to verify whether there exists a global correlation between them (i.e., whether there exist common patterns among the increasing and decreasing trends over time). To do so, we use the cross-correlation, which is a metric used in signal processing [77]. This correlation calculates the similarity between two time-series as a function of the displacement from one time-series to another. For example, given two functions f and g, the cross-correlation calculates the degree to which a shift in g (along the x-axis) is identical to the shift in f.

However, since we are also interested in analyzing the relationship between the peaks of our two time-series, we use the Dynamic Time Warping (DTW) [58], which was designed for comparing time-series varying at different speeds. A peak in *potential users'churn* may be followed by a peak in *issue reports* (or vice-versa), i.e., the peaks may not necessarily occur at the same time. Differently from the Eucledian distance, which assumes that a point i in a time-series must be aligned to the i_{th} point in another time-series [38, 75], DTW allows the variance in the offset between two time-series [59] .

To employ DTW, we use the `tslearn` [68], which is a machine learning toolkit specialized for time series analysis. The DTW computation produces a matrix that shows the distance between every two points in the graph of the two time-series. The algorithm is designed in a way to choose the closest point that respects the

Table 2: The predominant topics in alternativeto.net

Release and updates	
Keywords	"*version*", "*longer*", "*release*", "*download*", "*official*"
Examples	- The program seems to be no longer updated. Last versions (2018) can be still downloaded from the official website. - Firefox version 60+ (Quantum) is better than Chrome. Compare to previous versions of Firefox.
Bookmarks Functionality	
Keywords	"*bookmark*", "*panel*", "*favorite*", "*support*", "*sync*"
Examples	- Firefox now allows bookmarks importing from Chrome which is a huge plus. - With the recent upgrade to 2.0 (including the bookmarks syncing feature that I liked in Firefox), Vivaldi just got better.
Flash support	
Keywords	"*plugin*", "*video*", "*install*", "*flash*", "*media – player*"
Examples	- Since its the only way to watch Flash video on an iPhone without jailbreaking, Skyfire is a killer app thats worth getting for anyone. Playback of Flash video over WiFi is flawless, and the quality is, at very least, watchable.
Memory issues	
Keywords	"*memory*", "*RAM*", "*lag*", "*limit*", "*heavy*"
Examples	- Great C++ IDE which balances memory usage and indexing solution. Contains some basic refactoring functions. Good for large projects and limited RAM with remaining some capabilities of more complex C++ IDEs. - Eclipse is really lagging after the last update. The memory is filled whenever I run a Java swing applet. - Chrome is so heavy, it lags alot when I open multiple tabs.
Privacy and security	
Keywords	"*privacy*", "*security*", "*important*", "*allow*", "*control*"
Examples	- Firefox allows far more control over your privacy than does Chrome (or even Chromium on which Chrome is based). - Google Chrome is a great browser and a lot better than Vivaldi Browser. Chrome keeps updating with cool stuff like the new cast feature, better extensions and lot better privacy and security. Vivaldi has more customizing features, but has a long way to go to compete with Chrome.
Bugs/Crashes	
Keywords	"*crash*", "*bug*", "*fix*", "*experience*", "*issue*"
Examples	- Lots of users experience Adobe Flash Player hangs or crashes. I dont know why it isnt fixed yet. If Chrome ever got its proxy support bugs fixed it would leap to the top of the list and become pretty much the only web browser Id ever use. - I worked with NetBeans for a couple of years and always loved it, but for professional users the yearly fee is well spent: Nicer interface, better Code Intelligence, more options to fine tune what happens on saving a file (eg. saving or uploading to various locations). You get a couple of feature update each year plus bugfixes, whereas NetBeans is sometimes very slow to update things (support for new version of a language) or to fix known bugs.

Table 3: The predominant topics in alternativeto.net (continued)

Debug options	
Keywords	"*feature*", "*development*", "*debug*", "*integration*", "*perfect*"
Examples	- Not enough debug and run options; just a text editor, Atom has not got any solid debugging features (and packages) out of the box or at all. - This is my favorite code editor. It is open source and portable, you can get a portable version that will fit onto a flash drive if you want. It is just as configurable as Sublime Text and has a large community producing great plugins. it has GIT support built in and excellent support for Javascript, HTML, CSS, and Python, and others of course. It has an integrated debugging system.
Simplistic design	
Keywords	"*simple*", "*editor*", "*lightweight*", "*design*", "*notepad*"
Examples	- Really lightweight with some advanced, not complete but easy to use auto complete features. Simple design and customizable.
Cross-platform	
Keywords	"*cross – platform*", "*Linux*", "*open – source*", "*available*", "*install*"
Examples	- Its cross platform and open source and bring the benefits of the JetBrains IDE platform which is well polished and powerful - One of the best open source browsers that work on Linux and Windows.
Code refactoring and auto-completion	
Keywords	"*code*", ""*auto – completion*", "*refactor*", "*option*", "*use*"
Examples	- For its young age it is well matured and a very powerful IDE with great CMake Support and other goodies like great refactoring tools and a sophisticated auto-completion system.
Testing	
Keywords	"*test*", "*server*", "*deploy*", "*website*", "*set*"
Examples	- XAMPP is great tool to develop and test your website (particularly if it uses php, and mysql databases) offline before putting it on a live server. You could also use XAMPP as an easy method of setting up a live server. - Eclipse is far more superior than netbeans in unit testing functionality.
Documentation	
Keywords	"*documentation*", "*server*", "*port*", "*database*", "*forum*"
Examples	- Easiest way to run a web server on Windows. The performance is surprisingly good and lots of documentation on how to manage it. - Support is via forum only. Documentation can be very poor, particularly around product upgrades (one expects more from commercial packages). Due to their poor documentation, I lost several years of development databases when porting from an old to a new computer (they dont accurately document the location of the database files). - Eclipse lacks in adding plugins documentation.

time constraints for continuity and monotonicity. Finally, two co-authors performed a manual analysis of the observed peaks in the two time-series.[7] The goal of the manual analyses is to find whether the identified peaks are indeed likely related (e.g., the issue reports

[7] We share the data of our manual analyses at https://doi.org/10.5281/zenodo.3610584

Table 4: The extracted topics percentages distribution over the three domains

Topics	Browsers(%)	IDEs(%)	Web servers(%)
Release and updates	18	17	**21**
Bookmarks functionality	12	0	0
Memory issues	23	**22**	11
Privacy and security	8	0	0
Bugs/Crashes	**24**	19	20
Debug options	0	10	0
Simplistic design	0	8	2
Cross-platform	8	6	5
Code refactoring and auto-completion	0	10	0
Testing	0	2	17
Documentation	0	2	15
Miscellaneous	7	4	9

within a peak actually describe problems mentioned in the potential users'churn). Again, both co-authors analyze and discuss every analyzed sample.

In this RQ, we perform our analyses only in our studied projects (as opposed to analyzing all the software alternatives in their domains). As discussed in Section 3, it would be impracticable to extract the data from all the different Issue Tracking Systems of the software alternatives in the studied domains.

Findings: We respectively observe a cross-correlation of 0.76, 0.72, and 0.70 in Firefox, Eclipse and Apache. These cross-correlation scores indicate that there exist a global commonality between the trends of the two time-series regardless of the timing factor.

Regarding our DTW analysis, we observe that the time from the filing of an issue report to the time of a spark in potential users'churn span from 2 to 4 weeks (on the median for the three projects). Additionally, our DTW correlation reveals a bidirectional relationship between the peaks in issue reports and the peaks in potential users'churn. For the cases where a peak in issue reports leads to a peak in potential users'churn, it is intuitive to think that the peak in potential users'churn is due to the frustration generated by the issues described in the reports. On the other hand, the cases where a peak in potential users'churn is followed by a peak in issue reports may occur due to users being frustrated over non-obvious issues. For example, if we consider the potential user churn depicted in Figure 3 (i.e., *"Jupyter is slow on Firefox"*), the users might not find it obvious that such a problem should be reported to the Firefox team (e.g., users may simply interpret it as a characteristic of Firefox). Therefore, in such cases, issue reports would be filled only a while after the potential users'churn have been expressed.

We are specifically interested in studying the relationship between reported issues (i.e., the documented issues in the tracker systems) and the potential user churn. Our rationale is that understanding the characteristics of issues that are associated with user churn can help developers to prioritize such issues and avoid future churn (thereby, retaining more users).

Finally, Table 5 shows a subset of the examples found in our manual analyses. We can indeed observe the apparent relationship between the comments within the potential users'churn and the descriptions (or titles) of issue reports. The examples are following a unidirectional link, in which the issue report that occurs at a certain time would induce the potential user churn at a later time.

Summary: Our results suggest that there exists a significant correlation between software issues and potential user churn.

RQ2: Can we use machine learning models to predict the rate of potential user churn?

Motivation: In the practice, most of the issue reports undergo human estimations of the priority and the severity [5], usually based on internal business-oriented factors (such as ROI [25]). Issues might be prioritized by their recency, the affected platforms, the versions of the software where the issue is present. Feeding a machine learning model to identify the potential user churn that are caused by issues may help automate the issue report prioritization process and ultimately save huge costs in software development.

Approach: We train three different machine learning models to predict *low* and *high* user churn rates by using information from issue reports. The first model is a Feed-Forward Neural Network with two hidden layers developed by using the framework Keras [17]. Keras is widely used to train simple neural networks as it has a higher abstraction than TensorFlow [1]. Feed-Forward Neural Networks [64] are the most suitable type of neural networks to be trained on tabular data [27, 33, 50]. This network evaluates all the possible combinations of different metrics values to guide the predictions.

Our second model is the XGboost model, which is a gradient boosting tree [12, 13]. XGboost is widely used for binary classification problems. Research has shown that XGboost performs better than neural networks and classic classification machine learning algorithms in problems dealing with tabular data [14, 45, 69].

We also train Random Forest models [35]. Random Forest is a classical machine learning algorithm which is widely used in problems dealing with tabular data [26, 51, 57]. In addition, Random Forest models have been widely used in software engineering research, such as in defect prediction [67].

The goal of our models is to predict the rate of potential users'churn given the characteristics of issue reports. The data used to fit our models is the data obtained from our DTW associations. Simply put, the DTW provides us with P pairs of issue reports I and potential users'churn C (i.e., $P < I, C >$). Therefore, we study the characteristics of the issues I to predict the amount of potential user churn C. We split the distribution of C into two percentiles (i.e., *low* and *high*). The low percentile being from 0% to 50% and the high from 51% to 100%. Thus, our models output dichotomous predictions, i.e., whether the potential users'churn is *low* or *high*.

The characteristics of issue reports that we include in our models (henceforth referred to as *features*) are described in Table 6. These features are collected from the Issue Tracking systems of our three studied projects. We study features such as *Component*, *Hardware*, and *OS* to account for the locality and the spread of the issues. For example, these features can indicate whether platform-specific issues or more general issues have more potential user churn.

Other features, such as the *Severity* and *Priority* showcase the prioritization used by developers. The *Severity* and *Priority* are important for us to verify whether the prioritization performed by developers is aligned with the rate of potential user churn.

Features related to the type of the bugs (i.e., whether it is a crash), whether a version number is present, or the target milestone, provide us information regarding the tracking process of issues. They are important to understand whether better-tracked issues

Table 5: Manual investigation of DTW correlation linking

Firefox	
Example 1	
Correlation Linking	Week 164 in issues to week 169 in switches (May 2018)
Issue Report Title	Firefox version 53.0.3 appears high memory, high resources usage and might causing for SSD damages
Issue Description	Mozilla Firefox it is caching all of this data 4-5GB on my new SSD and its lost 4% from its health within 5 months, due continuously writes/overwrites firefox cache on my SSD.
Altrnativeto Comment	Switching to Chrome after version 53.0.3, high memory is damaging my SSD
Example 2	
Correlation Linking	Week 66 in issues to week 68 in switches (November 2016)
Issue Report Title	Sessions are not cleared in the private window
Issue Description	After closing private window (not a tab) and opening a new one I still logged in.
Altrnativeto Comment	Chrome incognito functionality is more stable than Firefox, Firefox doesn't resets sessions.
Eclipse	
Example 1	
Correlation Linking	Week 10 in issues to week 11 in switches (July 2015)
Issue Report Title	Not Java 8 Compatible
Issue Description	When downloading Eclipse for the first time for Java development, 64-bit (which is what my computer is), it is not running because it is not compatible with Java version 8. (which is the latest version of Java), but appears to be compatible with version 7 still -Dosgi.requiredJavaVersion=1.7
Altrnativeto Comment	Using Netbeans because Eclipse is not installing.
Example 2	
Correlation Linking	Week 211 in issues to week 213 in switches (February 2019)
Issue Report Title	jar files in the exported application do not contain class files
Issue Description	The application has many plugins. It builds and runs well within eclipse, but when I export the application using the eclipse export product wizard, the jar files (corresponding to the plugins) do not contain any class files! they include meta-inf, plugin.xml, and all the extra folders (eg lib, assets, etc, when available), but they do NOT contain any class files.
Altrnativeto Comment	Didn't find any problems in exporting jar (when recommending IntelliJ).
Apache	
Example 1	
Correlation Linking	Week 113 in issues to week 117 in switches (July 2017)
Issue Report Title	httpd 2.4.26 no longer building against lua 5.3.1 or lua 5.3.4
Issue Description	Building Apache 2.4.26 against lua 5.3.1, or 5.3.4 compiled from scratch (latest version as of this writing) does not work. Compilation against lua 5.3.1 did work with 2.4.25.
Altrnativeto Comment	Some issues while building with lua (when recommending XAMPP).
Example 2	
Correlation Linking	Week 12 in issues to week 13 in switches (September 2019)
Issue Report Title	Apache Server is restarted time to time (Release 2.4.10)
Issue Description	I am using Apache release 2.4.10 in Windows Server 2008. Time to time the Apache server is getting restarted. The service is unavailable for 3-5 minutes and after that the service is started up automatically. This issue occurs frequently. twice a day, two days once etc..
Altrnativeto Comment	XAMPP is more stable, I am experiencing random restarts throughout the day when using Apache.

Table 6: The issue reports attributes for Firefox, Eclipse and Apache

Metric	Description
Component	consists of different components of the system such as bookmarks, theme, toolbar, etc.
Status	describes the status of the issue such as unconfirmed, open, resolved, etc.
Resolution	describes the resolution type of the issue such as fixed, duplicate, invalid, etc.
Severity	the assigned severity such as blocker, critical, normal, etc.
Hardware	the different hardware affected by the issue such as desktop, mobile, 32 or 64 bits, etc.
OS	the different OS affected by the issue such as OSX, Windows, etc
Priority	the assigned priority from 1 to 5
Type	the assigned issue type such as defect, enhancement, or task
Time Alive	the time difference between the date opened and the date resolved in minutes
Time since last change	the time difference between the date opened and the date of last change in minutes
Is a Crash	a boolean value to indicate if it is a crash report
Has a Version Number	a boolean whether the issue report has a version number
Has a Target Milestone	a boolean whether the issue report has a target number
Has a Regression Range	a boolean value to indicate if the bug causes a regression testing
Number of comments	the number of comments for the report
Number of votes	the number of votes for the report, which validates the integrity of the report by other users
Number of Blocks	the number of issues that are blocked by that issue until it is resolved

(i.e., with more tracking information) are associated with potential users'churn.

The *number of comments*, *votes*, and *blocks* showcase the effort invested on discussing an issue. Finally, the *time-alive* and the *time-since-last-change* are two features related to the life-cycle of an issue report. Longer *time-alive* values indicate a lingering issue that has not been resolved yet, while a short *time-since-last-change* values indicate issues that have been recently addressed (and thus can still be reoccurring).

Our studied features are inspired by previous research in software engineering that have built machine learning models using information present in issue reports [15, 16, 16, 18, 19, 29, 32, 39, 74].

We chose features that are common across the issue reports of Firefox, Eclipse and Apache. We filtered out features that are unique to an issue tracker system only to ensure a fair comparison between our studied issue reports. Afterwards, we performed Spearman correlation tests [44] on the selected features, in which each feature is tested against the set of all the other features. We did not observe any correlation obtaining a value higher than 0.7, which suggests that our selected features are not correlated and can be safely fed into our machine learning models [30].

To evaluate the performance of our models, we use the *Area Under the Curve* metric (AUC). AUC is useful for evaluating our models because our models output probabilities. Therefore, AUC shows the discrimination power of our models in every probability threshold (unlike Precision, Recall, or F-measure, which are limited to evaluating models at a single probability threshold [66]). The AUC values range from 0 to 1. An AUC of 0.5 denotes a random guessing model while an AUC of 1 denotes a perfect distinguishing power. An AUC of 0 denotes a model with perfect inverse predictions.

Given the temporal nature of our data (i.e., future issue reports cannot be used to predict the potential user churn of past issue reports), we adopt a leave-one-out validation approach to obtain our AUC values.

First, our issue reports have already been sorted when we perform the time-series analyses. Second, we split our data into two

sets: one containing 80% of the data, i.e., the training set and another containing 20%, i.e., the validation set. In the first iteration of our models, we use 80% of the data to train our models and we use the *first element* of the validation set to test out models. Then, the leave-one-out validation iteratively increases the training set size by one element (from the validation set) and predicts the next element of the validation set. This process is repeated until the training set reaches the size of $N - 1$ (where N is the size of our data set). Finally, our obtained AUC values are the aggregation (i.e., median) of all the AUC values obtained in the leave-one-out iterations.

Findings: The AUC values obtained for our three models are shown in our online appendix.[8] The XGboost model performs the best in the three studied projects, obtaining AUC values that exceed 0.83. There exists a subtle difference between the AUC values of the XGboost model and the Feed Forward Neural Networks AUCs, since both the XGboost model and the Neural Networks can grasp the complex relationships in our issue reports data.

Summary: Our results suggest that machine learning models, such as XGboost and Feed Forward Neural Networks, can predict the rate of user churn with relatively high accuracies (in terms of AUC). Such predictions could help in the prioritization of issue reports.

RQ3: What are the most important factors related to potential user churn?

Motivation: In RQ2, we observe that machine learning models can be used to predict the rate of potential users'churn. However, developers would hardly blindly trust a machine learning model to help on their decisions (and they should not do so). Developers would mostly benefit from understanding the reasons behind the predictions of our machine learning models, so that they could possibly adapt (or not) their development process. Therefore, in RQ3, we investigate the most important features in the predictions of our machine learning model.

Approach: In this RQ, we study the most important features in the predictions of the XGboost models, since they obtain the best AUC values in our three studied projects.

To find the most important features, we adopt a simple *feature extraction algorithm* [61, 70]. Consider that our models are trained on a feature set $X = x_1, x_2, ..., x_n$ (as explained in RQ2). The feature extraction process consists of iteratively (i) removing each feature x_i from our models and (ii) computing the AUC without each feature x_i (by using the same leave-one-out validation process explained in RQ2). The AUC values from the models without a feature x_i are compared to the models containing all the features (i.e., we take the difference $AUC_{original} - AUC_{removed}$). The higher the drop in the AUC value caused by removing a certain feature x_i, the higher the importance of such a feature x_i [4, 9, 10].

Findings: Table 7 shows our obtained results after performing the feature extraction process. The table shows the drop in the AUC for each feature (i.e., ΔAUC). We also show the minimum, maximum, mean, and median values of the features in the *low* user churn rate (i.e., 0) and *high* user churn rate (i.e., 1) prediction classes.

[8] https://doi.org/10.5281/zenodo.3610584

The *Issue Alive Time* is the most important feature, obtaining the highest ΔAUC in our three studied projects. This result suggests that it is unhealthy to leave issues hanging for a long time as they can be perceived as a reason for users to churn.

The *Number of Comments* obtains a considerable ΔAUC in our three studied systems. The association between the *Number of Comments* and the rate of user churn may occur due to the multitude number of users being affected by an issue.

The *Time-since-last-change* is also an important feature in the Firefox and Eclipse projects. This result suggests that issues with recurring changes or modifications may signal a certain instability in the fixing process, which might cause a potential user churn in the future.

It is worthy noting that features, such as *Severity*, *Priority*, and *Type of Bugs* do not obtain a high importance in the predictions, which might suggest that the existing prioritization fields used in issue reports are misaligned with the potential user churn.

Summary: Our results suggest that long lived issues can potentially lead to more potential user's churn and that the existing processes for prioritizing issues should be augmented to capture the highly interactive and long lived issues.

5 THREATS TO VALIDITY

Construct Validity: The main construct validity of our study is related to the assumption of a *potential user churn*. The *alternativeto.net* website does not record exactly when a user has chosen a software alternative over another software. Instead, *alternativeto.net* records that a user *thinks* that a software may be a good alternative to another software. The motivation behind the act of signaling an alternative is not known by us. For example, there might be users that are simply driven by the passion of contributing to the *alternativeto.net* community, which lead them to provide several opinions as to which software could make good alternatives to others. We have deliberately adopted the term *potential user churn* instead of simply *user churn* due to this unknown motivations behind signaling alternatives. Moreover, it is reasonable to think that if a given software has received a considerable amount of recommendation for alternatives, this may likely indeed represent that users are considering to change the software.

Internal Validity: Our internal threats to validity are mainly related to (i) our chosen features and (ii) our prediction classes. We split the peaks of potential user churn into the *low* and *high* classes. Other research may find different results if the potential user churn is modeled in different way. In addition, we acknowledge that our set of metrics is not exhaustive. For example, we have not studied whether the presence of a stack-trace in an issue report can be related to future potential user churn. We plan to (i) model the potential user churn as a continuous variable and (ii) extend our set of features to predict the potential user churn in future research.

External Validity: Our study targeted three domains that are web servers, IDEs and web browsers. These three domains were chosen due to their availability, since we could fetch their issue reports data. Although our study showed common factors in our analyses, we cannot generalize our results to other domains or projects with a different scale (i.e., smaller projects). Regardless, our work provides insights of the possible reasons behind user churn.

Table 7: The most important features for Firefox, Eclipse, and Apache ranked by their drop in AUC

Factor	ΔAUC	Min 0	Max 0	Mean 0	Median 0	Min 1	Max 1	Mean 1	Median 1
Firefox									
Time Alive	0.13	0	120	11.02	13	10	1487	48.72	63
OS All	0.07	0	1	0.14	0	0	1	0.83	1
Number of Comments	0.06	1	65	8.94	12	1	458	28.11	45
Time Since Last Change	0.06	9	30	10.90	22	0	5	4.10	3
Has a Target Milestone	0.05	0	1	0.22	0	0	1	0.79	1
Has a Version Number	0.04	0	1	0.33	0	0	1	0.64	1
Component UI Web Payments	0.03	0	1	0.42	0	0	1	0.61	1
Eclipse									
Time Alive	0.12	0	103	9.01	12	8	1334	43.96	49
Has a Target Milestone	0.06	0	1	0.19	0	0	1	0.82	1
Time since last change	0.06	7	27	8.51	19	0	6	3.12	3
Has a Version Number	0.05	0	1	0.34	0	0	1	0.77	1
Component Debug	0.04	0	1	0.47	0	0	1	0.72	1
Number of Comments	0.03	1	72	10.92	11	1	632	27.01	72
OS All	0.02	0	1	0.50	0	0	1	0.68	1
Apache									
Time Alive	0.14	0	136	11.32	14	14	1521	53.10	70
Component Utils	0.10	0	1	0.22	0	0	1	0.91	1
Has a Version Number	0.09	0	1	0.38	0	0	1	0.86	1
Number of Comments	0.05	1	92	12.32	6	1	352	42.07	79
OS All	0.04	0	1	0.33	0	0	1	0.79	1
Status REOPENED	0.03	0	1	0.40	0	0	1	0.73	1
Hardware All	0.02	0	1	0.51	0	0	1	0.78	1

6 RELATED WORK

Given that we study the relationship between *potential user churn* and issue reports, our research is related to the areas of *user/customer churn* and *software quality*. Therefore, we survey the related research around these two areas in this section.

User/Customer Churn. User or customer churn has been widely studied in the area of telecommunication services [3, 28, 71]. Amin et al. [3] proposed a cross-company machine learning model to predict customer churn. The authors also propose the use of data transformation (e.g., by using the log function) to improve the training data. Ullah et al. [71] used feature selection techniques, such as Information Gain and Correlation Attributes Ranking Filter to select which features better explained the churn behaviour of different customer groups. The authors observed that a Random Forest model performed the best in their study. While the Business to Customers (B2C) churn has been widely studied, Figalist et al. [28] recently investigated the Business to Business (B2B) churn, i.e., when the customer business changes shifts to the services provided by the competition. Figalist et al. [28] allude that although B2B relationships tend to be more stable, they have a much bigger financial impact when they change. In terms of software engineering, there has been a lack of studies that have investigated user/costumer churn directly. Instead, much research has been invested on customer reviews regarding software products [6, 46, 47]. Noei et al. [46] studied which features from mobile apps are associated with better ranks in the Google Play store. While the work by Noei et al. [46] does not study the customer churn directly, striving for better ranks in Google Play may avoid user churn in mobile apps. Bavota et al. [6] investigated the relationship between the usage of fault- and change-prone APIs in Android apps and user reviews. Differently from all the aforementioned work, our study investigates the user churn is software products.

Software Quality. Although there exist a lack of studies analyzing the user churn of software products/services, there has been a considerable amount of research on software quality [5, 34, 39, 66, 67, 74]. Research on software quality is important, since a quality software is more stable and less defect-prone. Ultimately, such characteristics are tightly related to the observations of our study(i.e., software issues are correlated with user churn). A prominent area of software quality research in software engineering is the defect prediction area [66, 67]. Tantithamthavorn et al. [66] studied how much improvement can be obtained by automatically optimizing defect prediction models. Their research shows that automatic optimization can yield significant or insignificant performance gains depending on the machine learning algorithm. Other considerable amount of research has been invested on the triaging part of issue reports [5] and on the effort estimation of an issue report (in terms of time to address an issue) [39, 74]. In this paper, we complement prior research by studying the relationship shared between issue reports (e.g., defects or required enhancements) with the *potential user churn* of users.

7 CONCLUSION

In this work, we study the data available on *alternativeto.net* to better understand the relationship between software issues and the *potential user churn* of users. Having observed that user concerns are tightly related to software issues (e.g., bugs), we investigate the relationship between issue reports and the *potential user churn* of users. Our study reveals key issues that must be addressed for the success of a software product (depending on the domain). For example, we observe that the potential user churn of users may be tightly related to the lack of a robust documentation and support for testing tools (in the "IDE" and "Web Server" domains). Finally, our machine learning models reveal that (i) the longer the issue takes to be fixed, the higher the chances of user churn; and (ii) issues within more general software components are more likely to be associated with user churn. Finally, we suggest that the current prioritization performed by developers should be augmented to encompass the long lived and highly interactive issues. In overall, our study suggests that the prioritization process of issues can be improved by considering the potential user churn of users associated with such issues.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] W Abdelmoez, Mohamed Kholief, and Fayrouz M Elsalmy. 2012. Bug fix-time prediction model using naïve bayes classifier. In *2012 22nd International Conference on Computer Theory and Applications (ICCTA)*. IEEE, 167–172.

[3] Adnan Amin, Babar Shah, Asad Masood Khattak, Thar Baker, and Sajid Anwar. 2018. Just-in-time customer churn prediction: with and without data transformation. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–6.

[4] Hafeez Ullah Amin, Aamir Saeed Malik, Rana Fayyaz Ahmad, Nasreen Badruddin, Nidal Kamel, Muhammad Hussain, and Weng-Tink Chooi. 2015. Feature extraction and classification for EEG signals using wavelet transform and machine learning techniques. *Australasian physical & engineering sciences in medicine* 38, 1 (2015), 139–149.

[5] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. ACM, 361–370.

[6] Gabriele Bavota, Mario Linares-Vasquez, Carlos Eduardo Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. The impact of api change-and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering* 41, 4 (2014), 384–407.

[7] Pamela Bhattacharya and Iulian Neamtiu. 2011. Bug-fix time prediction models: can we do better?. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 207–210.

[8] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.

[9] Andrew P Bradley. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern recognition* 30, 7 (1997), 1145–1159.

[10] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[11] Albert Caruana and Michael T Ewing. 2010. How corporate reputation, quality, and value influence online loyalty. *Journal of Business Research* 63, 9-10 (2010), 1103–1110.

[12] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[13] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* (2015), 1–4.

[14] Wenbin Chen, Kun Fu, Jiawei Zuo, Xinwei Zheng, Tinglei Huang, and Wenjuan Ren. 2017. Radar emitter classification for large data set based on weighted-xgboost. *IET Radar, Sonar & Navigation* 11, 8 (2017), 1203–1207.

[15] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2015. Predicting delays in software projects using networked classification (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 353–364.

[16] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Aditya Ghose, and John Grundy. 2017. Predicting delivery capability in iterative software development. *IEEE Transactions on Software Engineering* 44, 6 (2017), 551–573.

[17] François Chollet. 2017. Keras.

[18] Daniel Alencar Da Costa, Shane McIntosh, Uirá Kulesza, Ahmed E Hassan, and Surafel Lemma Abebe. 2018. An empirical study of the integration time of fixed issues. *Empirical Software Engineering* 23, 1 (2018), 334–383.

[19] Daniel Alencar da Costa, Shane McIntosh, Christoph Treude, Uirá Kulesza, and Ahmed E Hassan. 2018. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering* 23, 2 (2018), 835–904.

[20] Piew Datta, Brij Masand, Deepak R Mani, and Bin Li. 2000. Automated cellular modeling and prediction on a large scale. *Artificial Intelligence Review* 14, 6 (2000), 485–502.

[21] Kushal S Dave, Vishal Vaingankar, Sumanth Kolar, and Vasudeva Varma. 2013. Timespent based models for predicting user retention. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 331–342.

[22] William H DeLone and Ephraim R McLean. 1992. Information systems success: The quest for the dependent variable. *Information systems research* 3, 1 (1992), 60–95.

[23] William H DeLone and Ephraim R McLean. 2003. Model of information systems success: a ten years update. *Journal of Management* (2003).

[24] Gideon Dror, Dan Pelleg, Oleg Rokhlenko, and Idan Szpektor. 2012. Churn prediction in new users of Yahoo! answers. In *Proceedings of the 21st International Conference on World Wide Web*. ACM, 829–834.

[25] Khaled El Emam. 2005. *The ROI from software quality*. Auerbach Publications.

[26] Katherine Ellis, Jacqueline Kerr, Suneeta Godbole, Gert Lanckriet, David Wing, and Simon Marshall. 2014. A random forest classifier for the prediction of energy expenditure and type of physical activity from wrist and hip accelerometers. *Physiological measurement* 35, 11 (2014), 2191.

[27] David Enke and Suraphan Thawornwong. 2005. The use of data mining and neural networks for forecasting stock market returns. *Expert Systems with applications* 29, 4 (2005), 927–940.

[28] Iris Figalist, Christoph Elsner, Jan Bosch, and Helena Holmström Olsson. 2019. Customer churn prediction in B2B contexts. In *International Conference on Software Business*. Springer, 378–386.

[29] Emanuel Giger, Martin Pinzger, and Harald Gall. 2010. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, 52–56.

[30] Frank E Harrell Jr. 2015. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer.

[31] Steffen Hedegaard and Jakob Grue Simonsen. 2013. Extracting usability and user experience information from online user reviews. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2089–2098.

[32] Yujuan Jiang, Bram Adams, and Daniel M German. 2013. Will my patch make it? and how fast?: Case study on the linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 101–110.

[33] Guolin Ke, Jia Zhang, Zhenhui Xu, Jiang Bian, and Tie-Yan Liu. 2019. TabNN: a universal neural network solution for tabular data. https://openreview.net/forum?id=r1eJssCqY7

[34] Foutse Khomh, Brian Chan, Ying Zou, and Ahmed E Hassan. 2011. An entropy evaluation approach for triaging field crashes: A case study of mozilla firefox. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 261–270.

[35] Andy Liaw and Matthew Wiener. 2002. Classification and regression by random forest. *R news* 2, 3 (2002), 18–22.

[36] Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi. 2007. Mining eclipse developer contributions via author-topic models. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 30–30.

[37] Xi Long, Wenjing Yin, Le An, Haiying Ni, Lixian Huang, Qi Luo, and Yan Chen. 2012. Churn analysis of online social network users using data mining techniques. In *Proceedings of the international MultiConference of Engineers and Conputer Scientists*, Vol. 1.

[38] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[39] Lionel Marks, Ying Zou, and Ahmed E Hassan. 2011. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. ACM, 11.

[40] Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. (2002).

[41] Miloš Milošević, Nenad Živić, and Igor Andjelković. 2017. Early churn prediction with personalized targeting in mobile social games. *Expert Systems with Applications* 83 (2017), 326–332.

[42] Audris Mockus, Roy T Fielding, and James Herbsleb. 2000. A case study of open source software development: the Apache server. In *Proceedings of the 22nd international conference on Software engineering*. Acm, 263–272.

[43] Gustavo Percio Zimmermann Montesdioca and Antônio Carlos Gastaud Maçada. 2015. Measuring user satisfaction with information security practices. *Computers & Security* 48 (2015), 267–280.

[44] Leann Myers and Maria J Sirois. 2004. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences* 12 (2004).

[45] Didrik Nielsen. 2016. *Tree boosting with XGBoost-why does XGBoost win" Every" Machine Learning Competition?* Master's thesis. NTNU.

[46] Ehsan Noei, Daniel Alencar Da Costa, and Ying Zou. 2018. Winning the app production rally. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 283–294.

[47] Ehsan Noei, Feng Zhang, Shaohua Wang, and Ying Zou. 2019. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering* 24, 4 (2019), 1964–1996.

[48] Ehsan Noei, Feng Zhang, and Ying Zou. 2019. Too Many User-Reviews, What Should App Developers Look at First? *IEEE Transactions on Software Engineering* (2019).

[49] Derek O'callaghan, Derek Greene, Joe Carthy, and Pádraig Cunningham. 2015. An analysis of the coherence of descriptors in topic modeling. *Expert Systems with Applications* 42, 13 (2015), 5645–5657.

[50] Ya A Pachepsky, Dennis Timlin, and GY Varallyay. 1996. Artificial neural networks to estimate soil water retention from easily measurable data. *Soil Science Society of America Journal* 60, 3 (1996), 727–733.

[51] Mahesh Pal. 2005. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing* 26, 1 (2005), 217–222.

[52] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Software maintenance and evolution (ICSME), 2015 IEEE international conference on*. IEEE, 281–290.

[53] Chitra Phadke, Huseyin Uzunalioglu, Veena B Mendiratta, Dan Kushnir, and Derek Doran. 2013. Prediction of subscriber churn using social network analysis. *Bell Labs Technical Journal* 17, 4 (2013), 63–76.

[54] Peter C Rigby, Daniel M German, and Margaret-Anne Storey. 2008. Open source software peer review practices: a case study of the apache server. In *Proceedings of the 30th international conference on Software engineering*. ACM, 541–550.

[55] Martin P Robillard and Robert Deline. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (2011), 703–732.

[56] Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*. ACM, 399–408.

[57] Victor Francisco Rodriguez-Galiano, Bardan Ghimire, John Rogan, Mario Chica-Olmo, and Juan Pedro Rigol-Sanchez. 2012. An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing* 67 (2012), 93–104.

[58] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (1978), 43–49.

[59] Stan Salvador and Philip Chan. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11, 5 (2007), 561–580.

[60] Peter B Seddon. 1997. A respecification and extension of the DeLone and McLean model of IS success. *Information systems research* 8, 3 (1997), 240–253.

[61] Rudy Setiono, Bart Baesens, and Christophe Mues. 2008. Recursive neural network rule extraction for data with mixed attributes. *IEEE Transactions on Neural Networks* 19, 2 (2008), 299–307.

[62] Dong-Hee Shin. 2015. Effect of the customer experience on satisfaction with smartphones: Assessing smart satisfaction index with partial least squares. *Telecommunications Policy* 39, 8 (2015), 627–641.

[63] Tom De Smedt and Walter Daelemans. 2012. Pattern for python. *Journal of Machine Learning Research* 13, Jun (2012), 2063–2067.

[64] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39, 1 (1997), 43–62.

[65] Shaheen Syed and Marco Spruit. 2017. Full-text or abstract? Examining topic coherence scores using latent dirichlet allocation. In *2017 IEEE International conference on data science and advanced analytics (DSAA)*. IEEE, 165–174.

[66] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 321–332.

[67] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* (2018).

[68] Romain Tavenard. 2017. tslearn: A machine learning toolkit dedicated to time-series data. https://github.com/rtavenar/tslearn.

[69] L Torlay, Marcela Perrone-Bertolotti, Elizabeth Thomas, and Monica Baciu. 2017. Machine learning–XGBoost analysis of language networks to classify patients with epilepsy. *Brain informatics* 4, 3 (2017), 159.

[70] Geoffrey G Towell and Jude W Shavlik. 1993. Extracting refined rules from knowledge-based neural networks. *Machine learning* 13, 1 (1993), 71–101.

[71] Irfan Ullah, Basit Raza, Ahmad Kamran Malik, Muhammad Imran, Saif Ul Islam, and Sung Won Kim. 2019. A churn prediction model using random forest: analysis of machine learning techniques for churn prediction and factor identification in telecom sector. *IEEE Access* 7 (2019), 60134–60149.

[72] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 14–24.

[73] Chih-Ping Wei and I-Tang Chiu. 2002. Turning telecommunications call details to churn prediction: a data mining approach. *Expert systems with applications* 23, 2 (2002), 103–112.

[74] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. 2007. How long will it take to fix this bug?. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 1–1.

[75] Ian H Witten, Eibe Frank, and A Mark. 2016. Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques* (2016).

[76] Barbara H Wixom and Peter A Todd. 2005. A theoretical integration of user satisfaction and technology acceptance. *Information systems research* 16, 1 (2005), 85–102.

[77] Jae-Chern Yoo and Tae Hee Han. 2009. Fast normalized cross-correlation. *Circuits, systems and signal processing* 28, 6 (2009), 819.