Web Frameworks for Desktop Apps: an Exploratory Study

Gian Luca Scoccia
gianluca.scoccia@univaq.it
DISIM, University of L'Aquila
L'Aquila, Italy

Marco Autili
marco.autili@univaq.it
DISIM, University of L'Aquila
L'Aquila, Italy

ABSTRACT

Background: Novel frameworks for the development of desktop applications with web technologies have become popular. These *desktop web app frameworks* allow developers to reuse existing code and knowledge of web applications for the creation of cross-platform apps integrated with native APIs. *Aims:* To date, desktop web app frameworks have not been studied empirically. In this paper, we aim to fill this gap by characterizing the usage of web frameworks and providing evidence on how beneficial are their pros and how impactful are their cons. *Method:* We conducted an empirical study, collecting and analyzing 453 desktop web apps publicly available on GitHub. We performed qualitative and quantitative analyses to uncover the traits and issues of desktop web apps. *Results:* We found that desktop web app frameworks enable the development of cross-platform applications even for teams of limited dimensions, taking advantage of the abundant number of available web libraries. However, at the same time, bugs deriving from platform compatibility issues are common. *Conclusions:* Our study provides concrete evidence on some disadvantages associated with desktop web app frameworks. Future work is required to assess their impact on the required development and maintenance effort, and to investigate other aspects not considered in this first research.

KEYWORDS

Web technologies, desktop apps, cross-platform.

ACM Reference Format:
Gian Luca Scoccia and Marco Autili. 2020. Web Frameworks for Desktop Apps: an Exploratory Study. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3382494.3422171

1 INTRODUCTION

In the modern Internet economy, reducing the friction experienced by users while using a product can have a significant impact on its success. For instance, Google found that 53% of mobile site visitors leave a web page if it does not load within three seconds [16]. One possible way to improve the user experience is to provide a dedicated app for each type of device available to the user to take full advantage of the capabilities offered by the underlying platform, i.e., providing a mobile and a desktop app in addition to a traditional web application.

In this context, similarly to what already happened for mobile applications [11], web frameworks dedicated to the development of desktop applications have recently emerged [6]. Specifically, these frameworks allow developers to create a desktop application by providing: (i) a headless web browser instance that runs the application logic and renders the user interface; (ii) JavaScript bindings for accessing native OS APIs from the web-based code. Hereafter, we will refer to desktop apps built with web frameworks as *desktop web apps*.

The desktop web app approach comes with multiple advantages: being contained within the browser, desktop web apps can be packaged and distributed over any platform supported by it; the access to native APIs enables the use of capabilities unavailable to standard web apps, e.g., desktop notifications, system tray; existing libraries and knowledge of web developers can potentially be reused for the development of desktop applications; only a single code base needs to be maintained, which can be distributed on all platforms, thus simplifying the development process. On the negative side, being executed within the browser, desktop web-apps might incur performance overhead, and JavaScript bindings are provided for only a subset of all the possible APIs that exist on each platform.

Thanks to the advantages they offer, desktop web apps have attracted considerable interest. *Electron* is the prominent desktop web app framework with hundreds of thousands of weekly downloads[1] and hundreds of apps have been developed with it, including popular applications such as WhatsApp, Slack, Skype, Atom, and Visual Studio Code[2]. In a recent survey of 21,717 JavaScript developers, more than 70% declared that they have already used desktop web app frameworks or have an interest in using them in the future [5].

Until now, despite their growing popularity, desktop web apps have not been studied empirically: the how desktop web app frameworks are used in practice has not been investigated yet. Moreover, there is only anecdotal evidence on how their pros are beneficial and how impactful are their cons, when considering the trade-offs they impose on developers. As a step forward, we present an empirical study that characterizes their usage and investigates traits and issues of desktop web apps, by mining and analyzing 453 desktop web apps publicly available on GitHub. Specifically, from the analysis of collected data, we provide insights into the kinds of desktop applications that are developed employing web frameworks and

ESEM '20, October 8–9, 2020, Bari, Italy
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7580-1/20/10…$15.00
https://doi.org/10.1145/3382494.3422171

[1]https://www.npmjs.com/package/electron
[2]https://wiredelta.com/10-most-popular-electron-apps-2019/

the typical size of their development team. In addition, by extracting data from the manifest file of each application, we assess how common is the reuse of web libraries and what libraries are mostly reused. Finally, we study how common are platform compatibility issues for desktop web apps, by manually classifying issue reports collected from GitHub.

The main findings of our study are: (i) desktop web app frameworks are mainly used for the development of tools and utilities, while less commonly used for the development of entertainment apps; (ii) the majority of desktop web apps is developed by a small development team, that hardly would be able to support a native app for each targeted platform; (iii) web libraries are frequently reused for the development of desktop web apps; (iv) plugins that provide functionalities not offered by desktop web app frameworks are also commonly used; (v) platform compatibility issues are a frequent occurrence for desktop web apps.

2 WEB FRAMEWORKS FOR DESKTOP APPS

At the time of writing, two main desktop web app frameworks exist and are actively maintained: *Electron* and *NW.js* [6]. Electron is an open-source framework developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript. Electron accomplishes this by combining Chromium and Node.js into a single run-time. Its development began in 2013, and it was open-sourced in the Spring of 2014. NW.js (formerly node-webkit) is a framework for building desktop web apps with HTML, CSS, and JavaScript. NW.js achieves this purpose by combining together the WebKit web browser engine and Node.js. It has originally been created in 2011 by the Intel Open Source Technology Center.

The main difference between the two frameworks is the way they implement the integration between the Node.js back-end and the browser front-end: while NW.js maintains a single state shared between the two, Electron keeps a separate state for the back-end process and the front-end app window. Despite this and other small differences, the two frameworks are comparable in potential and both allow for the creation of feature-rich applications.

3 STUDY DESIGN

3.1 Goal and research questions

The primary **goal** of the study is to investigate existing traits and issues of desktop web apps, to characterize their usage and their communities. The **context** of the study is 453 open-source desktop web apps collected from GitHub. We refined this goal into the following research questions:

RQ1 What kind of applications are developed by employing desktop web app frameworks?

RQ2 What is the size of the development team of a desktop web app?

RQ3 To what extent are existing web development libraries reused in desktop web apps?

RQ4 How common are platform compatibility issues for desktop web apps?

RQ1 and RQ2 are of exploratory nature. Specifically, the purpose of RQ1 is to understand how desktop web app frameworks are used in practice and whether these are considered general-purpose by developers, or adopted only for some restricted purposes. RQ2 investigates the nature of the typical development team that uses desktop web app frameworks, since some of the benefits promised by the latter are specifically aimed at small teams with limited resources (e.g., cross-platform compatibility, skill reuse). RQ3 and RQ4 instead aim to provide empirical evidence on some of the promised benefits of desktop web app frameworks. Indeed, reusing web libraries is one of the ways in which developers can reuse existing code and the skills they already possess to develop desktop applications. Hence, answering RQ3 provides an indication of how common reuse is and what is commonly reused. Cross-platform compatibility is another of the promised benefits, but its effectiveness might be limited if numerous platform compatibility issues occur during development. RQ4 focuses on this aspect.

3.2 Data collection and extraction

As starting point for the data collection, we considered the two lists of user-submitted apps published on the NW.js and Electron official website. Indeed, both websites provide a list of apps developed with the corresponding framework to showcase its capabilities and to be used as a reference by developers. Both lists are open and anyone can freely add his app to the list. We collected both lists as of the 5th of February 2020, containing 998 Electron and 138 NW.js applications, respectively. From these, by means of ad-hoc scripts, we removed the ones that do not provide a link to a working GitHub repository, leaving us with 528 Electron and 66 NW.js apps. When working with GitHub repositories, there is a risk of including inactive or abandoned repositories and unfinished applications into our study [7]. In order to mitigate this risk, we considered only (i) repositories containing at least 10 commits, and (ii) with a span of at least 8 weeks between first and last commit in the repository. A total of 405 Electron apps and 48 NW.js apps survived this filtering step. We then cloned the GitHub repository of each app and collected from each the following:

- *Commit Log Details.* Using Git's *commit log*[3], we retrieved data associated with each commit, such as the committer, the commit timestamp, and the commit message.
- *Github repository metadata.* For each repository, the number of stars, watchers, issues, and commits was collected from the repository web page.

[3]https://git-scm.com/docs/git-log

Table 1: Demographics of apps included in the study (SD = standard deviation, IQR = inter-quartile range)

		Min	Max	Median	Mean	SD	IQR
	Commits	11	37,550	201	962.95	2,853.35	554
	Committers	1	578	4	18.64	51.34	10
Electron	**Stars**	0	52,256	114	1,248.95	3,756.72	592
	Watchers	0	2,504	9	43.69	151.42	28
	Issues	1	15,455	27	294.03	1,144.96	139
	Commits	13	1,139	123.5	258.1	304.47	334.5
	Committers	1	65	4	6.69	10.14	4
NW.js	**Stars**	3	6,080	93.5	511.73	1,114.65	476.25
	Watchers	1	218	12.5	25.85	39.59	31.5
	Issues	1	1,228	32	106.69	229.93	97.5

- The *package.json* file. An associated metadata file that, following the structure common to JavaScript packages [19], contains, among others, version information, keywords associated with the app, and the list of dependencies.
- *Issue reports*. An issue report is a request for improvements, bug fixes, or the addition of new features [3]. For each issue report, we collected the title, its author, the labels assigned to it, its current status (i.e., open or closed), the creation date and the last edit date.

Repositories included in the dataset contain 362,223 and 11,559 total commits, made by 7,551 and 321 distinct committers, for Electron and NW.js apps respectively. Demographics of apps included in the study are summarized in Table 1. The median number of commits for Electron (NW.js) apps in our study is 201 (123.5), with a median number of 4 (4) committers per app, a median number of 114 (93.5) stars for projects, and a median number of 9 (12.5) watchers per project, respectively. Based on these numbers, we are reasonably confident that the apps considered in our study are adequately representative of real-world projects. A total of 108,379 and 6,331 issue reports were collected from Electron and NW.js repositories, respectively.

3.3 Analysis

In order to provide an answer to RQ1, we relied on descriptive statistics, computing counts and percentages for each app category for Electron apps in our dataset. We collected the category of each app (as declared by its developer) from its listing on the framework official website. In answering this research question, we exclude NW.js apps since for these apps the category is not available (developers are not required to specify the category in its listing).

Answering RQ2 required to identify the core developers of the app among all project committers. In order to perform this task, we extracted the list of all committers from the project commit logs as first, and then we applied the heuristic proposed by Kouters et al. to deal with the *aliasing problem*, i.e., the issue of developers using multiple identities when committing on the same repository [8]. According to this heuristic, two committers are considered the same developer if they share the same email prefix, i.e., the part before the @ symbol and their identities can be merged accordingly. Despite its simplicity, the Kouters' heuristic has proven effective in empirical evaluation [18]. After merging duplicated identities, we applied the commit-based heuristic of Yamashita et al. [21], so as to identify the project core developers. According to this heuristics, the core developers of a project are the ones whose combined contributions amount to 80% of all the project contributions.

Answering RQ3 required to extract the web development libraries employed by each project from its *package.json* file, which includes a list of all project dependencies. We performed the extraction by using an ad-hoc script. We then computed the usage count of each library and manually assessed the purpose of the most widely adopted libraries.

We answered RQ4 by manually identifying platform compatibility issues within issue reports. Towards this goal, we first performed an additional selection of the collected issue reports by discarding the ones written in a language different from English. For this selection, we relied on the Google Translate language detection API[4], resulting in a final number of 104,349 issues collected for Electron apps and 4,800 for NW.js ones. Afterward, we resorted to manual classification of the issue reports by following the guidelines of descriptive coding [15]. We employed random sampling to obtain samples of our data reasonably sized for manual analysis, fixing a 99% confidence level and 5% confidence interval. This resulted in the selection of two samples of 661 issues from Electron projects and 585 from NW.js projects. To reduce possible bias, two different researchers independently analyzed both complete samples. After completing the analysis, the two researchers aligned the labels with each other, solved all the cases in which there was a disagreement, and grouped similar labels. We make use of the Krippendorff's Alpha [10] to measure agreement between the two, before solving disagreement cases. We selected this measure for its ability to adjusts itself to small sample sizes. We obtained an $\alpha = 0.874$ over the two combined samples. Values of α over 0.8 are considered as an indication of reliable agreement [9].

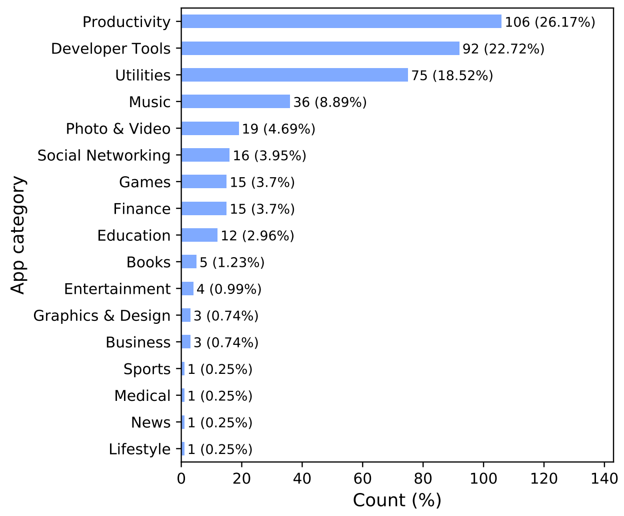

Figure 1: Electron apps by category

4 RESULTS

This section presents and discusses the results for each research question.

RQ1: What kind of applications are developed by employing desktop web app frameworks?

The plot in Figure 1 provides an overview of the distribution of desktop web apps across categories. The three most popular categories in our dataset are *Productivity* (26.17%), *Developer Tools* (22.72%) and *Utilities* (18.52%). Combined together, these categories account for over two-thirds of all applications, revealing that desktop web app frameworks are mainly used for the creation of apps and tools related to the professional sphere; whereas, they are less commonly used for entertainment apps. This is not surprising if considering the trend of the last few years, which sees other platforms gaining

[4]https://cloud.google.com/translate/docs/advanced/detecting-language-v3

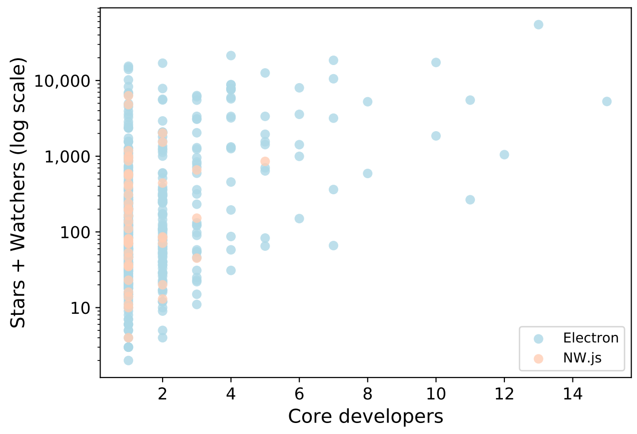

Figure 2: Distribution of core developers over project popularity

a more central role regarding entertainment (e.g., mobile devices), relegating the desktop to the workplace [20].

Among the remaining categories, *Music* (8.89%) and *Photo & Video* (4.69%) are the most numerous ones, highlighting that multimedia related apps are relatively common within desktop web apps. The variety of the remaining less numerous categories shows that desktop web app frameworks are indeed general-purpose and allow for the creation of apps with the most diverse scopes and extents. Of relevance, it must be noted that the *Medical* category is one of the least popular ones, with a single application belonging to it. This means that desktop web apps are rarely considered by developers for this kind of application. We believe this is due to the fact that, in order to effectively and correctly operate, this type of application generally requires data related to user habits or collected by sensors. Therefore, desktop apps are at a disadvantage when compared to other solutions, notably mobile apps, which can more easily collect this information.

RQ2: What is the size of the development team of a desktop web app?

Figure 2 shows the distribution of core developers for each project in our dataset in relation to its combined sum of GitHub stars and watchers that we use as a proxy for its popularity. As it can be seen, the number of core developers is mostly very limited and only sporadically it exceeds ten units. Descriptive statistics confirm this trend, with a median number of 1 core developer for both Electron and NW.js apps, and standard deviation of 1.95 and 1.38, respectively. We can also observe that the number of core developers tends to be growing as the popularity of an application grows. Nonetheless, it remains quite limited for the apps in our dataset, with a maximum value of 15 core developers for Electron apps and 6 for NW.js ones. Therefore, in light of these considerations, we can say that desktop web app frameworks are mostly used by small development teams, which would hardly be able to develop a native app for each of the targeted platforms.

RQ3: To what extent are existing web development libraries reused in desktop web apps?

The distribution of the number of included libraries for Electron and NW.js apps are provided by the boxplots of Figure 3. The median number of libraries included in the projects of our study is 7, but a considerable difference can be observed between NW.js apps (median value of 2) and Electron ones (median value of 8). However, in the more extreme cases, such number can drastically increase, with a maximum of 29 included libraries for NW.js and 153 for Electron apps. This suggests that the reuse of existing libraries is indeed common and becomes essential in the case of more complex applications.

Table 2 shows the 25 packages more frequently used in our combined dataset of 453 desktop web apps. We can observe that, except for some Electron specific plugins, the rest are libraries for traditional web development. Among these, we note the presence of multiple libraries (*react*, *jquery*, *vue*) that aid in the creation of user interfaces. In addition, the list includes web libraries for simplifying multiple aspects of web-app development: state management (*redux*), file manipulation (*fs-extra*), HTTP requests (*request*, *axios*), manipulation and displaying of dates (*moment*). Interestingly, we can observe that several plugins implement features missing in desktop web app frameworks: data persistence (*electron-store*), logging (*electron-log*), and windows management (*electron-window-state*). From this observation, we can evince a twofold conclusion: if on the one hand desktop web apps make considerable reuse of the large number of available web libraries, on the other, they often do it to remedy the lack (or inadequacy) of features offered by desktop web app frameworks. Indeed, some features, such as file manipulation and data persistence, are more relevant when developing desktop apps than in traditional web development. Therefore, we believe that desktop web app frameworks can benefit from the inclusion of some of these features. We believe that this is a crucial point that deserves more attention, and we plan to investigate this aspect more thoroughly in future work. The idea is to collect feedback directly from developers of desktop web apps, as described in Section 7.

RQ4: How common are platform compatibility issues for desktop web apps?

The bar plot in Figure 4 shows the results of our manual classification procedure. Four macro-categories emerged during the manual labeling of the issue reports, namely, *Bug report*, *Feature request/suggestion*, *Technical support request*, and *Documentation request*. The meaning of the first two categories is self explanatory. The third one groups all those reports where a user explicitly requests technical support (e.g., *"How to run in Ubuntu?"*). The last

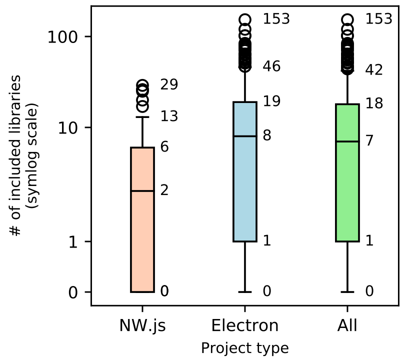

Figure 3: Distribution of included libraries by project type

category groups all the reports which request improvements to the project documentation (e.g., *"Update examples/documentation"*). A total of 1,044 distinct issue reports were classified in these four categories; the remaining 206 issue reports were excluded from our classification as either the report title did not provide enough information to clearly collocate the issue in the categories (e.g., *"About Update to v0.6.5"*), or the discussed topics were not strictly related to the repository project (e.g., *"J5 Collaborator Summit at JSConf US"*). Indeed, it is well known that GitHub issues are oftentimes used to discuss topics not related to software maintenance [1]. The distribution of issues across the categories is as follows. *Bug report* is the most numerous category, with 585 issue reports belonging to it, followed by *Feature request/suggestion* with 342. Less populous are *Technical support request* and *Documentation request* categories with 88 and 29 issues, respectively.

During our analysis, we identified 126 issues as being platform-related, i.e., making the relation to a specific platform explicit in the title. Among these, the vast majority (104 out of 126) belong to the category *Bug report* (e.g., *"Koala isn't working on Ubuntu 15.04"*), hence reaching a considerable percentage (17.78%) of all bug reports. These issues instead represent a more limited share of the total in categories *Feature request/suggestion* (e.g., *"Auto Update feature for Mac"*) and *Technical support request* (e.g., *"I need help to do this work fine in linux"*), with a total share of 4.97% and 5.68%, respectively. Unsurprisingly, they are completely absent in the category *Documentation request.*

Platform compatibility issues appear to be unevenly distributed among the two frameworks and across applications. Indeed, in our manual analysis, we identified 69 platform compatibility issues over 30 NW.js projects, while 57 were identified over 114 Electron projects. Descriptive statistics confirm this observation, with a median value of 1 (3.16 standard deviation) for NW.js, and a median value of 0 (1.52 standard deviation) for Electron. The maximum number of occurrences of this issue kind in a single project is 13 and 14 for NW.js and Electron, respectively.

Although our analysis is preliminary, in light of the fact that almost one in five bugs is platform-related, we believe that platform compatibility issues are a rather frequent occurrence during the development of a desktop web app. We plan to investigate more in-depth the impact of these bugs on the development process in future work (see Section 7). Nonetheless, we recommend desktop web app framework developers to increase their efforts to limit the occurrence of these bugs.

Table 2: Most used packages (top 25)

#	Package	# apps	#	Package	# apps
1	react	101	14	semver	39
2	react-dom	98	15	prop-types	39
3	electron-updater	66	16	electron-window-state	37
4	lodash	64	17	electron-is-dev	35
5	request	61	18	vue	33
6	moment	58	19	auto-launch	33
7	redux	54	20	react-router-dom	32
8	react-redux	51	21	axios	32
9	jquery	49	22	redux-thunk	32
10	fs-extra	48	23	electron-debug	30
11	electron-log	42	24	bootstrap	28
12	electron-store	42	25	font-awesome	28
13	uuid	41			

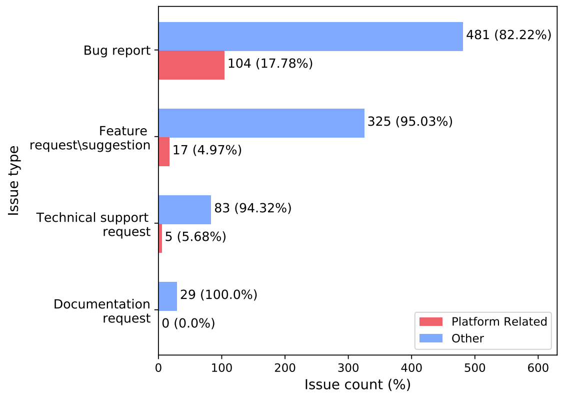

Figure 4: Analyzed issues by type

5 LIMITATIONS AND THREATS TO VALIDITY

Apps considered in our study are exclusively open-source and relatively limited in number. Hence, there is a risk that the results of our study might not generalize to the broader population of all desktop web apps. To mitigate this threat we have included in our study both main desktop web app frameworks and considered only apps of reasonable quality. Moreover, there is a risk that some proof of concept or unfinished applications have survived the filtering steps employed to avoid their inclusion (described in Section 3.2). Furthermore, in future work, we plan to replicate and extend the scope of our study by integrating other data sources (e.g., GH Torrent [4]).

In our study, a manual procedure was employed to identify platform compatibility issues, described in Section 3.2. As with all kinds of manual processes, mistakes might have occurred. To mitigate this threat, two different researchers performed this task independently. Their agreement level was measured with the Krippendorf alpha [10] and resulted satisfactory. During the data collection described in Section 3.2, we did not collect all the comments that appear in the discussion of each issue report. This choice was made to avoid incurring in the request limit imposed by the GitHub API. Hence, during the manual classification, we only relied on the issue title to classify each issue and we did not assign a label to the more subtle ones. Hence, our counts of issues in each category are to be considered as lower bounds of actual data.

6 RELATED WORK

To the best of our knowledge, this is the first study specifically aimed at exploring web frameworks for desktop apps. Thus, in this section, we relate to studies that, although driven by different objectives, share similarities with ours.

The idea of employing the browser as a platform for the execution of cross-platform applications is not novel. Indeed, in 2008, Taivalsaari and colleagues reported their experience in using a regular web browser as a platform for desktop applications [17], during which they encountered, among others, issues pertaining to platform fragmentation, performance, and inadequacy of the bindings between the browser and other system components. In a subsequent work [14], the same authors, discuss the ever-narrowing boundary between the web and desktop applications and the challenges that this trend poses for software engineering.

In the mobile domain, *hybrid mobile apps* allow developers to use web technologies for the development of their mobile application and distribute them in multiple app stores in a cross-platform fashion, leveraging techniques similar to those of desktop web apps. Malavolta et al. investigated the traits and the presence of hybrid mobile apps on the Google Play store [13]. In successive research, they focused on the differences between hybrid and native mobile apps as perceived by end users [12].

In the literature, some studies have empirically investigated the differences in bugs and bug-fixing processes over multiple platforms. Zhou and colleagues compared bug reports for three cross-platform projects that provide both a mobile and a desktop application [23]. Among their findings, they report that desktop bugs are less severe but on average require a longer time to be fixed. Zhang et al. compared the textual features of bug reports for popular desktop and mobile apps publicly available on GitHub [22]. They found that the average fixing time for desktop software was longer than that of mobile apps likely due to less descriptive bug reports.

7 CONCLUSIONS AND FUTURE WORK

We conducted an empirical study on desktop web app frameworks by mining and analyzing 453 open-source applications available on GitHub. The study provides concrete evidence on the use of such frameworks and the associated disadvantages. As next step, on the one hand, we will investigate more deeply the problems that emerged from this first research and, on the other, we will explore other aspects that have not been currently considered.

Specifically, in order to deepen and confirm the results of our study, in addition to replicating our analysis on a larger dataset (e.g., GH Torrent [4]), we plan to collect feedback directly from desktop web app developers, by means of direct interviews or through a questionnaire. Moreover, we want to investigate the impact of platform-related issues on development times, by analyzing the median time required to solve these issues and comparing it to the one necessary for issues of a different kind. More on the qualitative side, we also plan to manually analyze the code of issue-fixing commits in order to understand and characterize how the related problems were solved.

Regarding the aspects unexplored in this study, we want to investigate other potential criticalities of desktop web app frameworks. For instance, beyond bugs and security issues, being executed within a web browser, apps may suffer performance degradation and excessive energy consumption, especially if not properly optimized. Towards investigating these aspects, we plan to define and conduct laboratory experiments, which will allow us to collect measurements of performance metrics and energy consumption by employing specific tools and analyzers[5].

Finally, on a longer time frame, we will focus our attention on Progressive Web Apps [2], a technology that promises seamless cross-platform compatibility for web applications and whose support for desktop apps has recently been announced by Google[6] and Microsoft[7].

[5]https://github.com/GoogleChrome/lighthouse
[6]https://developers.google.com/web/updates/2018/10/nic70#dpwa-windows
[7]https://developer.microsoft.com/en-us/windows/pwa/

REFERENCES

[1] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2008. Is it a bug or an enhancement? A text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. 304–318.

[2] Andreas Biørn-Hansen, Tim A Majchrzak, and Tor-Morten Grønli. 2017. Progressive web apps: The possible web-native unifier for mobile development. In *International Conference on Web Information Systems and Technologies*, Vol. 2. SCITEPRESS, 344–351.

[3] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. 2013. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 188–197.

[4] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, Piscataway, NJ, USA, 233–236. http://dl.acm.org/citation.cfm?id=2487085.2487132

[5] Sacha Greif and Raphaël Benitte. 2019. *The state of JavaScript*. https://2019.stateofjs.com/mobile-desktop

[6] Paul B Jensen. 2017. *Cross-platform Desktop Applications: Using Node, Electron, and NW. js*. Manning Publications Co.

[7] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071.

[8] Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark GJ Van Den Brand. 2012. Who's who in Gnome: Using LSA to merge software repository identities. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 592–595.

[9] Klaus Krippendorff. 2004. Reliability in content analysis: Some common misconceptions and recommendations. *Human communication research* 30, 3 (2004), 411–433.

[10] Klaus Krippendorff. 2018. *Content analysis: An introduction to its methodology*. Sage publications.

[11] Ivano Malavolta. 2016. Beyond native apps: web technologies to the rescue!(keynote). In *Proceedings of the 1st International Workshop on Mobile Development*. 1–2.

[12] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. 2015. End users' perception of hybrid mobile apps in the google play store. In *2015 IEEE International Conference on Mobile Services*. IEEE, 25–32.

[13] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. 2015. Hybrid mobile apps in the google play store: An exploratory investigation. In *2015 2nd ACM international conference on mobile software engineering and systems*. IEEE, 56–59.

[14] Tommi Mikkonen and Antero Taivalsaari. 2011. Apps vs. open web: The battle of the decade. In *Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development*. MSE Santa Monica, CA, 22–26.

[15] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[16] Alex Shellhammer. 2016. The need for mobile speed: How mobile latency impacts publisher revenue. *DoubleClick by Google* (2016).

[17] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. 2008. Web browser as an application platform. In *2008 34th Euromicro Conference Software Engineering and Advanced Applications*. IEEE, 293–302.

[18] Igor Scaliante Wiese, José Teodoro Da Silva, Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2016. Who is who in the mailing list? Comparing six disambiguation heuristics to identify multiple addresses of a participant. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 345–355.

[19] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. 2016. A look at the dynamics of the JavaScript package ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 351–361.

[20] Yoram Wurmser. 2019. US Time Spent with Mobile 2019. *EMarketer. Last modified May* 30 (2019).

[21] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E Hassan, and Naoyasu Ubayashi. 2015. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution*. 46–55.

[22] Tao Zhang, Jiachi Chen, Xiapu Luo, and Tao Li. 2017. Bug Reports for Desktop Software and Mobile Apps in GitHub: What's the Difference? *IEEE Software* 36, 1 (2017), 63–71.

[23] Bo Zhou, Iulian Neamtiu, and Rajiv Gupta. 2015. A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. android vs. ios. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. 1–10.