Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub

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

Patrizio Migliarini
DISIM, University of L'Aquila
L'Aquila, Italy
patrizio.migliarini@graduate.univaq.it

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

***Abstract*—Software companies have an interest in reaching the maximum amount of potential customers while, at the same time, providing a frictionless experience. Desktop web app frameworks are promising in this respect, allowing developers and companies to reuse existing code and knowledge of web applications to create cross-platform apps integrated with native APIs. Despite their growing popularity, existing challenges in employing these technologies have not been documented, and it is hard for individuals and companies to weigh benefits and pros against drawbacks and cons.**

In this paper, we address this issue by investigating the challenges that developers frequently experience when adopting desktop web app frameworks. To achieve this goal, we mine and apply topic modeling techniques to a dataset of 10,822 Stack Overflow posts related to the development of desktop web applications. Analyzing the resulting topics, we found that: i) developers often experience issues regarding the build and deployment processes for multiple platforms; ii) reusing existing libraries and development tools in the context of desktop applications is often cumbersome; iii) it is hard to solve issues that arise when interacting with native APIs. Furthermore, we confirm our finding by providing evidence that the identified issues are also present in the issue reports of 453 open-source applications publicly hosted on GitHub.

***Index Terms*—Web technologies; Desktop apps; Stack Overflow; GitHub; Topic modeling.**

I. Introduction

A current challenge for business enterprises, software companies, and independent developers is to choose the target platforms for their applications. To reach a greater amount of users, companies, and developers aim at releasing their products on the largest manageable amount of platforms. More targeted platforms directly translate into a larger amount of potential customers and, thus, increased possible revenue and increased chances of product success [1].

In this context, similarly to what already happened for mobile applications [2], web frameworks dedicated to the development of desktop applications have recently emerged [3], [4]. 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. Henceforth, we will refer to desktop apps built with web frameworks as *desktop web apps*.

Desktop web app frameworks can potentially simplify application development, by enabling reuse of existing code and skills, while at the same time allowing to target multiple platforms [4]. However, the readiness level of these technologies is hard to assess as, currently, there is only anecdotal evidence on how their pros are beneficial and how their cons are impactful when considering the trade-offs they impose on developers. In this paper, we conduct an empirical study to precisely understand the needs and desiderata of desktop web apps developers. The ultimate aim is to clearly identify challenges and pain points experienced by developers, as well as possible aspects that framework maintainers can improve.

Towards this aim, we gather a data set composed of Stack Overflow posts in addition to a data set of GitHub issues. On the collected data, (i) we leverage topic modeling to understand the topics that are being discussed by web app developers on Stack Overflow and analyze them; (ii) we confirm our findings by replicating the topic modeling procedure on GitHub, evidencing that the identified challenges are also present in reported issues, hence also mitigating external validity risks.

Inspired by the work in [5], we targeted Stack Overflow and applied topic modeling to its posts since it is the most active question and answer site for programmers where to vote, up or down, challenging questions, reported issues, and provided solutions about the most disparate aspects in software development [6], [7]. Specifically, we leveraged topic modeling techniques to identify a number of coherent and meaningful topics from 10,822 Stack Overflow posts. Then, we apply quantitative and qualitative analyses to the identified topics, through both metrics and manual analysis, and employ different statistical tests. Numbers and dates related to the posts of interest for our study (reported in Section III-B) confirm the usefulness of Stack Overflow. Repeating the process on GitHub, we confirm our findings by providing evidence that the identified issues are also present in the issue reports of 453 open-source applications. Indeed, hosting over 4 million repositories [8], GitHub represents a useful source of information and has previously been used in a plethora of software engineering studies [9]. The target audience of our study is composed of desktop app developers and framework maintainers. Our study provides guidance to the former, as they can leverage our results to make a more informed decision when choosing the technologies for their projects. Our study benefits maintainers, evidencing a number of frameworks' pain points, on which they can direct their efforts.

The main findings of our study can be summarized as follow:

- developers often experience issues regarding the build and deployment processes for multiple platforms;
- reusing existing libraries and development tools in the context of desktop applications is often cumbersome;
- it is hard to solve issues that arise when interacting with native APIs.

In order to allow independent verification and replication of the performed study, we make publicly available a full replication package containing the obtained raw data and all the scripts employed for data preparation and analysis[1].

II. Web frameworks for desktop 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.

At the time of writing, two main desktop web app frameworks exist and are actively maintained: *Electron* and *NW.js* [3]. 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.

III. Study design

A. Goal and research questions

The **goal** of the study is to examine what desktop web apps developers are asking about, with the ultimate purpose of identifying challenges and pain points experienced during development, as well as possible aspects that framework maintainers can improve.

We refined this goal into the following research questions:

RQ1 What topics related to desktop web app development do developers ask questions about?
RQ2 Which topics are the most difficult to answer?
RQ3 How prevalent are difficult topics in issue reports of desktop web app development?

By answering RQ1, we want to identify the aspects of desktop web app development for which developers frequently ask for help and, hence, are commonly problematic for them. RQ2 instead aims to identify the ones that are the most difficult to answer within the previously detected topics and so can pose a problem for all but the most experienced developers. As done in [5], [10], [11], [12], in this paper we use the time it takes for an answer to be accepted across the different topics as a possible measurement to estimate difficulty. RQ3 builds on RQ2 and investigates the presence and proportion of the identified topics in real-world projects. Topic prevalence is a possible measurement to estimate the impact that most difficult topics have on the developments of desktop web apps.

B. Data collection

As illustrated in Figure 1, we started by gathering two main data sets that will constitute our research basin, composed of Stack Overflow posts and GitHub issues.

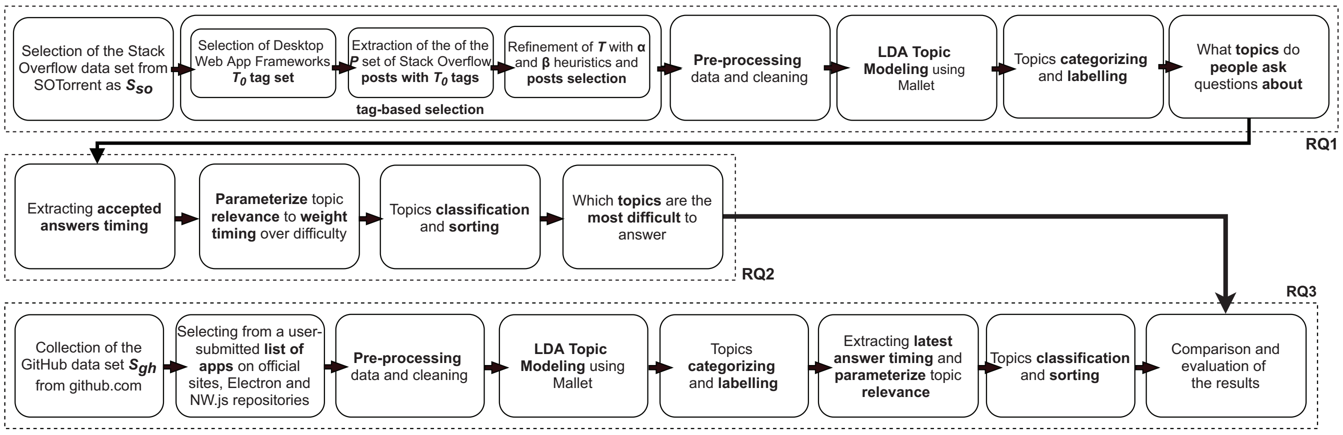
1) Stack Overflow Dataset: For the Stack Overflow dataset, we leveraged the *SOTorrent* dataset [13], an open data set based on the official Stack Overflow data dump. We employed the latest available version of the dataset, released in November 2020. This data set includes all questions present on Stack Overflow as of the 8th of September 2020, complete with their answers and related meta-data (hereafter referred to as S_{so}). The meta-data for each question includes the question's assigned tags (from one to five), the submission date, its view count, the question score, its favorite count, and – if present – the identifier of the answer that was marked as accepted by its original writer.

In order to make a selection of posts of interest for our research, we manually analyzed several Stack Overflow posts related to the Electron and NW.js frameworks. From this analysis, we defined an initial set of desktop web app related question tags T_0 = <*electron, nwjs, nw.js, node-webkit*>. Afterward, we extracted the question set P, composed of questions in S_{so} that contain at least one of the tags in T_0. From the latter, we defined T, the set of tags of posts in P.

Using notions from [14], [10] and [12] about *significance* and *relevance*, we calculated the significance heuristic α and the relevance heuristic β with the following formulas:

$$\alpha = \frac{\text{\# of posts with tag } t \text{ in } P}{\text{\# of posts with tag } t \text{ in } S_{so}} \quad \beta = \frac{\text{\# of posts with tag } t \text{ in } P}{\text{\# of posts in } P}$$

[1]https://github.com/gianlucascoccia/MSR2021Replication



Selection of the Stack Overflow data set from SOTorrent as S_{SO}
Selection of Desktop Web App Frameworks T_0 tag set
Extraction of the of the P set of Stack Overflow posts with T_0 tags
Refinement of T with α and β heuristics and posts selection
tag-based selection
Pre-processing data and cleaning
LDA Topic Modeling using Mallet
Topics categorizing and labelling
What topics do people ask questions about
RQ1
Extracting accepted answers timing
Parameterize topic relevance to weight timing over difficulty
Topics classification and sorting
Which topics are the most difficult to answer
RQ2
RQ3
Collection of the GitHub data set S_{gh} from github.com
Selecting from a user-submitted list of apps on official sites, Electron and NW.js repositories
Pre-processing data and cleaning
LDA Topic Modeling using Mallet
Topics categorizing and labelling
Extracting latest answer timing and parameterize topic relevance
Topics classification and sorting
Comparison and evaluation of the results

Fig. 1. Study design diagram

The heuristic α measures the relevance of a tag $t \in T$ to desktop web app development, while β measures the significance of a tag $t \in T$. We consider a tag t to be significantly relevant if its α and β values are higher than or equal to predetermined thresholds. Akin to previous work [14], [12], [10], we experimented with different values for the thresholds, and we found that the best results are achieved with $\alpha = 0.1$ and $\beta = 0.01$. Hence, employing the two heuristics, we refined T by keeping tags that are significantly relevant to desktop web app development, resulting in a final tag set $T =$ <*node-webkit*, *nw.js*, *nedb*, *electron*, *electron-packager*, *spectron*, *electron-builder*, *nwjs*, *electron-forge*>.

Extracting from S_{so} questions that possess at least one of the tags in T, we end up with a total number of 10.822 Stack Overflow posts. Demographics of resulting posts are provided in Table I. Questions with the *electron*, *electron-packager*, *spectron*, *electron-builder*, and *electron-forge* tags were considered related to the Electron framework, while questions with the *node-webkit*, *nw.js*, and *nwjs* tags were considered related to NW.js. Questions with the *nedb* tag have been considered for both frameworks, as the tag relates to a database technology that can be employed with either. Included Electron questions have a median number of 224 views (372 for NW.js) with a median number of 1 answer per question (1). The mean number of comments for each question is 1.43 and 1.35 for Electron and NW.js, respectively. The oldest Electron question in the dataset was asked in May 2014, while the oldest NW.js question dates back to November 2012. The newest question dates back to March 2020 for both frameworks. Based on these numbers, we believe that desktop web apps are sufficiently discussed on Stack Overflow, and therefore the selected questions can be a useful source of insights.

2) GitHub Dataset: We leveraged the two lists of user-submitted apps published on the NW.js and Electron official website for the collection of our GitHub dataset. Indeed, both websites provide a list of apps developed with the corresponding framework to showcase their capabilities and be used as a reference by developers. The lists are open, and anyone can freely add his app to the list. We collected both lists as of the 5th of February 2020, and - employing ad-hoc scripts - we filtered out apps that do not provide a link to a working GitHub repository, leaving us with 528 Electron and 66 NW.js apps. As it is common when working with GitHub repositories, there is a risk of including inactive or abandoned repositories and incomplete applications in the dataset [15]. 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 the first and last commit in the repository. A total of 405 Electron apps and 48 NW.js apps survived this filtering step. We then collected from each GitHub repository all available *issue reports*. An issue report is a request for improvements, bug fixes, or the addition of new features [16]. For each issue report, we collected its title, the full text of all posts on the issue discussion page, its author, the labels assigned to it, its current status (i.e., open or closed), the creation date, and the last edit date. The dataset's repositories contain 362,223 and 11,559 total commits, made by 7,551 and 321 distinct committers, for Electron and NW.js apps, respectively. 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)

TABLE I
DEMOGRAPHICS OF QUESTIONS IN THE STACK OVERFLOW DATASET (SD = STANDARD DEVIATION, IQR = INTER-QUARTILE RANGE)

		Min	Max	Median	Mean	SD	IQR
	Score	-8	296	0	1.52	5.88	2
	Views	4	130,805	224	1,079.1	3,977.89	666
Electron	**Answers**	0	19	1	0.93	0.94	1
	Comments	0	28	0	1.43	2.3	2
	Favorites	0	125	1	1.87	4.06	1
	Score	-6	151	1	1.56	6.32	2
	Views	5	82,022	372	1,095.69	3,804.39	847
NW.js	**Answers**	0	11	1	1.07	0.89	0
	Comments	0	22	0	1.35	2.16	2
	Favorites	0	58	1	1.92	3.95	1

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.

C. Data extraction

In the following, we describe the steps undertaken to extrapolate from the two datasets the information that we use to answer the three research questions

Pre-processing – We carried out some pre-processing steps, to clean up and prepare the collected data for the subsequent steps. We extract from all the documents of our datasets (i.e., Stack Overflow posts and GitHub issues) the respective titles, which we will use for our analysis. Indeed, titles have been found to be representative of the full document content and contain lesser noise that can skew the results of our analysis [5], [17], [18]. Afterward, we perform *stopwords removal*, i.e., the process of removing words commonly used in the English language, such as 'is', 'of', 'at' which do not significantly affect the semantics of a sentence and can potentially introduce noise. We leverage the NLTK stopwords [19] list for this operation. Subsequently, we perform *stemming*, the process of reducing inflected or derived words to their root form, and *lemmatization*, a process that reduces a word to its canonical form named lemma taking into consideration the linguistic context of the term (e.g., the word 'good' is the lemma of the word 'better').

Topics identification – To identify topics present in our datasets, we resort to topic modeling using the *Latent Dirichlet Allocation* (LDA) [20] algorithm, widely used in software engineering studies [21]. LDA is based on the idea that a "topic" consists of a cluster of words that frequently occur together in documents. LDA provides as output a series of probabilities for each document, representing the likelihood of a post being related to each of the identified topics. For our study, we employ the Mallet tool [22] implementation of LDA. LDA requires an input parameter K, representing the number of topics to search for. Determining the optimal K value is crucial for the analysis' results, as if its value is too small or too high, the algorithm might return topics that are respectively too narrow or too broad to yield any useful conclusions from. To overcome this challenge, we rely on the topics *coherence*. The coherence is one of the metrics commonly used to evaluate topic models and has been found to be highly correlated with human understandability [23]. For this purpose, we experiment with different values of K, ranging from 10 to 50 in increments of 5 and compute the mean coherence metric across all output topics. Selecting the K value with better coherence, we repeat the process in increments of 1 for values in the range $[K - 5; K + 5]$ and take note of the three values that provide the best results. We then pick the final value for K either by selecting the candidate value with the best coherence metric or, in cases were multiple values had similar scores, manually examining a sample of 50 documents for each candidate value. This procedure is similar to the one employed in [5] but it considers a wider starting range for possible values of K.

Naming topics – For the purposes of our analysis, it is necessary to understand the rationale that unites the documents in each topic identified by the LDA algorithm and to summarize it with a descriptive name. This step was performed by the first author, who has experience in JavaScript development. Subsequently, assigned topic names were revised and confirmed by the other two authors. To assign the topic names, the first author relied on the list of the top 20 words most frequently occurring in each topic, computed by the Mallet tool, and, when necessary, manually inspecting the 25 most relevant documents for a topic (i.e., the ones with the greater probability of belonging to the topic).

Calculating difficulty – We decided to take into account the relevancy of a GitHub issue GHi or a Stack Overflow question SOq to the selected topic t generated by LDA modeling while calculating its difficulty D. Thus, we parameterized the LDA probability P in the question/issue difficulty. As done in [5], [10], [11], [12], we calculate the difficulty subtracting question/issue Submit Timestamp (ST) from question Acceptance Timestamp (AT) for Stack Overflow and issue Closing Timestamp (CT) for GitHub as follows:

$$D(SOq_t) = (AT_{SOq} - ST_{SOq}) * P(SOq_t)$$

$$D(GHi_t) = (CT_{GHi} - ST_{GHi}) * P(SOq_t)$$

In other words, the $D(SOq_t)$ formula weights the difficulty of a question over its relevancy to a topic t. Indeed, a Stack Overflow question may include aspects related to multiple topics, e.g., a bug that manifests itself only on some platforms is to be considered related to both the *Errors* and the *Platform compatibility* topics. Hence, to take into account this multi-faceted nature of questions, in our analysis, we compute multiple $D(SOq_t)$ values for each question, each instantiated over a different topic. Analogous considerations are valid for the $D(GHi_t)$ formula.

D. Data analysis

To answer RQ1, we perform a qualitative analysis of the results of the topic modeling process. For each identified topic, we manually examine the top words produced for it by the LDA algorithm and a number of the most relevant questions encompassed by it. In this way we understand the shared rationale for the questions related to the topic and derive further insights.

To provide an answer to RQ2, we analyze collected data quantitatively. First, for each topic, we investigate the *normality* of the $D(SOq_t)$ distribution by employing the Anderson-Darling test [24], where the null hypothesis is that the data comes from a normal distribution. As we could always reject the null hypothesis, we adopt the omnibus Friedman test [25] to statistically determine if the weighted answer time for documents across identified topics exhibits a significant difference. The Friedman test is a non-parametric test for one-way repeated measures analysis of variance by ranks.

We use the Friedman test because collected data does not adhere to the assumptions of the ANOVA statistical test and the Friedman test is a non-parametric alternative that does not assume independence of observations [25]. We execute post-hoc analysis performing pairwise comparisons among each pair of topics employing Nemenyi's test [26]. The latter is a conservative test that accounts for family-wise errors, thus not requiring correction for obtained p-values [27].

To answer RQ3, we employ the LDA algorithm and the aforementioned tests (Anderson-Darling, Friedman, and Nemenyi) analogously to how these have been utilized to answer RQ1 and RQ2.

IV. Discussion on Findings

In this section, we list and discuss the results obtained from our analysis, broken down per research question.

A. RQ1: What topics related to desktop web app development do developers ask questions about?

Following the steps described in Section III-C, we obtained the best results for the LDA algorithm on the Stack Overflow dataset employing $K = 14$ topics. After reviewing the automatically generated topics, we manually merged a pair of semantically similar ones, leaving us with a final total of 13 topics. The resulting topics are displayed in Table II, alongside a selection of the top words for each topic, picked from the top 20 words automatically produced by the Mallet tool. Obtained topics are heterogeneous, covering several aspects (e.g., app architecture, tools, and the user interface) and phases of application development (e.g., design, testing, and deployment).

1) Topics overview: in the following, we describe and discuss the emerged topics in detail, making use of examples selected from the Stack Overflow questions that compose each topic. For reasons of space, in the following, we focus exclusively on the most relevant topics, while discussion and examples of the others can be found in the online appendix included in the replication package[1].

App architecture: questions in this topic discuss the fundamental logical structure of the desktop application in terms of, e.g., routes, views, and components. An example of this kind of questions is *"AngularJS $routerProvider not working properly in node-webkit"*, in which one developer asks for help in configuring the AngularJS router included in his application. Noticeably, the names of several JavaScript frameworks appear among the top words of this topic (as can be seen in Table II). Indeed, during JavaScript application development, it is common practice to adopt such frameworks to properly structure the application architecture when the logic becomes more extensive and difficult to maintain [28].

- Desktop web application development requires the usage of abstractions and frameworks to properly manage the application's logical structure when dealing with growing application complexity.

Build & deploy: this topic comprises questions about the build process of desktop web apps, whose ultimate goal is to create build artifacts that can be distributed and executed on multiple platforms. An instance of this type of question is *"What are some mechanisms to package cross-platform Electron apps in a single build?"*. The presence of this topic among the most discussed ones is, at a first glance, contrasting with one of the main touted strengths of desktop web app frameworks: the possibility of developing an application in a single language while still being able to easily distribute it on multiple platforms [3]. To investigate the matter more in-depth, we decided to conduct a manual analysis of questions relevant to this topic. From it, we noted that indeed developers require clarifications on these subjects even though desktop web app frameworks are designed to simplify deployment on multiple platforms due to the fact that developers often have specific requirements for the deployment of their applications on some platforms (e.g., *"How to deploy an Electron app as an executable or installable in Windows"*) or necessitate to include native libraries in their product and thus have to follow more elaborate build processes (e.g., *"Unable to load some native node js modules with electron 4.0.6 on Windows"*).

- Despite frameworks' efforts to simplify deployment across multiple platforms, developers often ask for help regarding the build and deploy processes.

Client-server: this topic groups questions asking for clarification regarding interactions between the desktop web app and a remote server. For instance, in the post *"Electron: socket.io can receive but not emit"* a developer states that he is *"creating an Electron application that uses Socket.io to communicate to a server application"* and asks for help in troubleshooting issues that arise when forwarding messages from one of the clients to the server. The presence of this topic reveals that desktop web apps are often not developed in isolation but serve as an (additional) client-side interface for existing applications and services. This is in line with one of the main advantages offered by desktop web app frameworks, namely the possibility of reusing the already possessed web development skills to develop desktop applications.

- Desktop web applications are often developed as an additional front-end client for existing applications and services.

Dependencies: this topic collects questions dealing with issues related to the inclusion of libraries or other software dependencies. An example is the post *"Requiring node modules in ionic + electron (5.0.0) desktop application"*. The ability to reuse existing web development libraries for desktop applications is advertised as one of the major strengths of desktop web app frameworks [3]. Hence, we deemed it appropriate to investigate the reasons behind the presence of this topic among the most discussed on Stack Overflow. Conducting a manual analysis of related questions, we identified two main reasons: firstly, as specified in the official Electron

TABLE II
TOPICS IN THE STACK OVERFLOW DATASET
(TOPICS IN ITALIC ARE IN COMMON WITH THE GITHUB DATASET)

Topic	Top words	Median $D(SOq_t)$ (minutes)	$\sigma(D(SOq_t))$ (minutes)
App architecture	react angular component function vue data change callback	16.39	16,847.37
Build & deploy	build packag creat electron-build electron-packag webpack exe bundl	21.32	29,934.15
Client-server	server request node.js client proxy connect express response	9.87	27,695.19
Databases	data nedb databas store sqlite set valu updat	8.89	14,033.18
Dependencies	modul requir node import nativ defin typescript angular	17.9	26,887.64
Errors	undefined error typeerror javascript uncaught empty null result	5.53	9,301.1
File manipulation	file save imag local download path open read	20.67	21,746.9
Inter-process communication	process render main window child ipc send communic	8.38	9,669.71
Developer tools	code variable javascript global function object source debug	6.42	12,103.8
Page contents	load page dom html webview script element tag	16.44	20,989.03
Platform integration	chrome print detect device memory screen shell python	20.31	23,279.03
Testing	test spectron run selenium browser working testing chrome	10.2	14,401.30
User interface	window show menu click browser screen close button	22.88	37,399.92

faq[2], the way these frameworks integrate the node.js backend and the frontend browser instance can result in compatibility issues when employing some popular libraries (e.g., JQuery or AngularJS), which require additional setup steps to be correctly integrated; secondly, it is common practice in the JavaScript ecosystem to use dependency managers, i.e., software libraries that assist in the integration of multiple external libraries. Integrating these within desktop web apps is not always straightforward. In both cases, solving these issues requires manually tweaking configuration files of libraries, frameworks, or underlying components (e.g., configuration files of the node.js backend server). Required edits are mostly specific for each library, hence deep knowledge of the involved technologies is necessary. For instance, the answer to the question *"Error: Can't resolve 'electron-is-dev' in electron & typescript & webpack project*" reports the need to configure the `webpack.config.js` file in order to integrate Electron with the Webpack module bundler.

- Reuse of traditional web development libraries within desktop web apps is common, but their integration is not always straightforward.

Developer tools: these are questions asking for explanations on how to use existing development tools, such as code editors and debuggers, in the context of desktop web apps. An example is the Stack Overflow post *"How to debug Quasar Electron App with VS Code"*. Similarly to what has been observed for the *Dependencies* topic, by manually analyzing questions related to the topic, we found that some of the tools commonly used by developers (e.g., IDEs, debuggers) require additional configuration steps or workarounds to be used for desktop web app development. One example is in the answer to the question *'Debug typescript electron program in vscode"* in which, to enable the usage of the IDE built-in debugger within the Electron application, the necessary edits to multiple IDE and build process configuration files are described.

- Some commonly adopted developer tools (e.g., debuggers, IDEs) cannot be used out-of-the-box for desktop web application development.

Platform integration: this topic aggregates those questions in which the developer asks how to invoke native APIs (e.g., *"node-server-screenshot not working on live ubuntu server"*) or how to interact with hardware peripherals (e.g., *"Accessing USB devices from node-webkit?"*). This topic is of primary importance, given that integration with the underlying platform is one of the main advantages offered by desktop web app frameworks. Manual exploration of related questions reveals that developers often face difficulties when their application needs to support multiple platforms, as not all APIs and behaviors are standardized across platforms. One example is given in the *"ELECTRON: image file(.png) silent printing on Ubuntu"* Stak Overflow post, where the accepted answer points out the need to employ two different APIs to implement printing of documents on Windows and Ubuntu. Moreover, developers often experience difficulty in integrating the required software libraries to bridge between the web application and the underlying platform. This stems from the fact that existing Node.js native modules cannot be used as-is but needs to be recompiled before usage, as desktop web app frameworks employ a different application binary interface[3][4].

- Developers face difficulties when supporting multiple platforms due to: (i) inconsistent APIs across platforms and (ii) difficulties in integrating native modules into the desktop web application.

Testing: these posts discuss aspects related to application testing, often seeking clarification regarding test frameworks and tools. *"Mocha test setup to run two tests who require same beforeEach setup"* is an example. Analyzing the questions of this topic, we noticed that the main reason why developers experience testing-related difficulties is that commonly used testing frameworks and tools are often not compatible with desktop web app frameworks. Instead, ad-hoc tools or wrappers for existing ones must be utilized in their place. Multiple examples are found in Stack Overflow questions: packages such as `spectron`, `electron-chromedriver` and `nw-chromedriver` provide wrappers for the popular ChromeDriver automated testing tool; whereas, `nw-test-runner` and `electron-mocha` wrap

[2]https://www.electronjs.org/docs/faq

[3]https://www.electronjs.org/docs/tutorial/using-native-node-modules
[4]https://www.npmjs.com/package/nw-gyp

around the Mocha testing framework.

• Ad-hoc wrappers are required to make existing testing frameworks and tools usable for desktop web application development.

2) Additional considerations: in the following, we provide some additional considerations on the presented results.

Reuse is possible but cumbersome – A common thread that binds several of the topics described above is the difficulty that developers encounter in reusing familiar technologies in the context of desktop web apps: libraries and testing frameworks frequently require workarounds or ad-hoc solutions to be employed, while tool support is lagging. This is a direct consequence of the technical solutions employed by current desktop frameworks to enable communication between the node.js back-end and the front-end browser window but also suggests that library and tool developers do not consider desktop web apps a potential target for their products. In other words:

• Despite their growing popularity, desktop web apps are still unaccounted for by many libraries, frameworks, and tools hence complicating their adoption alongside familiar technologies.

Skills required to develop desktop web apps – Another important aspect to consider is the possibility for developers to reuse, in addition to libraries and tools, skills already possessed for the development of desktop applications. From this point of view, analyzing the list of topics identified, we can indeed identify topics that encompass skills in common with traditional web development, such as *Page contents*, *Client-server* and *Databases*. However, we can also point out some topics that relate to skills less commonly used in traditional web development (i.e., *File manipulation* and *Inter-process communication*) in addition to others that are exclusive to desktop applications, as in the case of *Build & deploy* and *Platform integration*. Therefore, we recommend developers interested in using desktop web app frameworks to deepen their knowledge of these aspects.

• In addition to traditional web development skills, developers interested in desktop web applications should study in deep aspects such as *File manipulation*, JavaScript *Inter-process communication*, *Build & deploy* processes and APIs for *Platform integration*.

Evolution of desktop web app questions – In addition to the qualitative considerations previously provided, we have analyzed quantitatively the evolution of questions related to desktop web app frameworks over the years, displayed in Figure 2. Questions were plotted on the graph according to their creation date and divided into Electron- and NW.js-related questions on the basis of the tags assigned with the same procedure used in Section III-B1.

Examining the plot, we can observe that the combined number of questions is increasing over the years, likely due to a growing interest in desktop web app frameworks. More in detail, we can notice that starting from the second half of 2015, the number of questions related to Electron has continued to grow while the number of questions related to NW.js has slowly declined, widening the gap between the two frameworks. This highlights that, to date, Electron is by far the most popular among the two frameworks, even though NW.js maintains a presence within the developer community. We hypothesize that some peculiarities probably made Electron gain more popularity over the years: Electron does not require Chromium customization, it does not introduce a new JS context in pages, it receives latest security updates, it has a bigger community, more in-production apps using it, and more userland modules available in npm [29]. In addition, we can observe a spike in the number of monthly questions happening at the beginning of 2019, although future work is required to identify the reasons behind it.

• The number of monthly Stack Overflow questions discussing desktop web app development is experiencing a growing trend, mostly driven by an increasing interest in the Electron framework.

B. RQ2: Which topics are the most difficult to answer?

To identify the most difficult topics, we used the $D(SOq_t)$ measure, previously defined in Section III-D. Figure 3 provides a logarithmic scale boxplot of the $D(SOq_t)$ for each topic identified in the previous research question, while median and standard deviation values are reported in Table II. We can notice that the median $D(SOq_t)$ ranges from a minimum of 5.53 minutes for the topic *Errors* to a maximum of 22.8 minutes for the topic *User interface*. However, the standard deviation is very large for each topic ($\approx$15.5 hours minimum and 26 days maximum) signifying that, within each topic, the time taken to answer questions is very spread. Additionally, we observed small differences in value for the first quartile over the topics distributions (minimum $\approx$30 seconds, maximum $\approx$2 minutes) but large differences for the third quartile (minimum $\approx$40

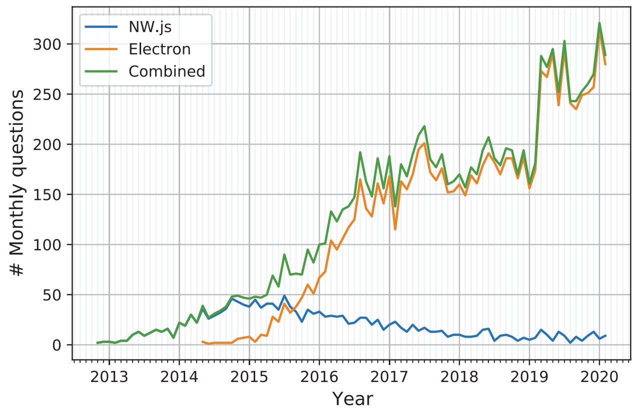


Fig. 2. Evolution of desktop web app questions on Stack Overflow over time

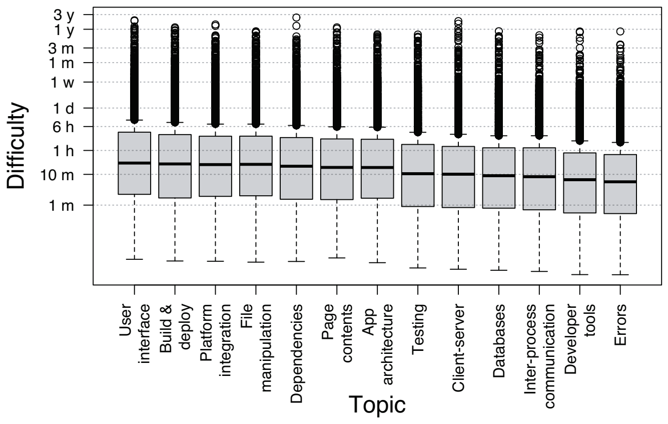


Fig. 3. Boxplot of $D(SOq_t)$ (difficulty of Stack Overflow topics)

minutes, maximum ≈4 hours), suggesting that the differences are more significant in the upper half of the distributions.

Differences in descriptive statistics suggest that the distribution of answer times differs across topics. We statistically test this hypothesis by applying the Friedman omnibus test. The result ($p-$value < 0.01) allows us to reject the null hypothesis that distributions of $D(SOq_t)$ across topics are not statistically significantly different. Subsequently, as post-hoc analysis, we tested the hypothesis that each topic has a statistically significantly greater distribution than the others. For this purpose, we performed pairwise comparisons in a round-robin fashion, employing the one-tailed Nemenyi's test. Based on the number of comparisons for which we were able to reject the null hypothesis, we sorted the topics according to their difficulty, obtaining the ranking shown in Figure 3. In all comparisons for which we were able to reject the null hypothesis (i.e., that the pivot topic distribution is not statistically significantly greater), we always obtained a $p-$value < 0.01.

From the obtained sorting, we notice that the most challenging topic (i.e., the one with a greater median $D(SOq_t)$) is *User interface*. This highlights a pain point in using desktop web app frameworks and therefore we recommend their developers to pay attention to the ease-of-use of APIs and mechanisms for the management of the user interface. Afterward, in order of difficulty, we find the topics *Build & deploy* and *Platform integration*, thus suggesting that integration and distribution on multiple platforms are problematic for developers. To these follow the topics *File manipulation* and *Dependecies*, confirming the criticalities discussed in the previous research question. Looking at the easier topics we find instead *Inter-process communication*, *Developer tools* and *Errors* signifying that for these aspects developers' queries are more rapidly solved, suggesting a minor impact on development times.

• *User interface*, *Build & deploy*, and *Platform integration* are the most difficult topics. *Inter-process communication*, *Developer tools* and *Errors* instead are the ones answered more rapidly.

C. RQ3: How prevalent are difficult topics in issue reports of desktop web app development?

To answer this question, we search for topics identified in RQ1 inside the GitHub dataset. For it, we obtained the best results for the LDA algorithm employing K = 13 topics. Also in this case, we merged a pair of automatically produced topics that were found to be semantically similar during the manual review. The resulting 12 topics are displayed in Table III, together with a selection of words for each topic selected from the top 20 words automatically produced by the Mallet tool. Due to space considerations, we omit the detailed discussion of each individual topic, available in the online appendix[1].

1) Topics overview: as in the case of Stack Overflow, obtained topics are reasonably heterogeneous. However, differently from it, we can note the presence of some topics more loosely connected to software engineering. We believe this is due to the fact that i) in the GitHub dataset there are multiple applications with functionalities closely related to these topics (e.g., messaging apps or cryptocurrency wallets); ii) it is well known that GitHub issues are oftentimes used to discuss topics not related to software maintenance [30].

We report that 6 out of 12 identified topics (i.e., *Build & deploy*, *Errors*, *File manipulation*, *Platform integration*, *Testing* and *User interface*) are also present in the Stack Overflow dataset, providing an evidence that the problems associated with these topics are frequently encountered during desktop web app development.

• The topics *Build & deploy*, *Errors*, *File manipulation*, *Platform integration*, *Testing*, and *User interface* are frequently found in issue reports of real-world GitHub applications.

2) Statistical analysis: Table III reports the median and standard deviation of the $D(GHi_t)$ measure for each topic identified in the GitHub dataset. We can observe that *Feature request* is the topic with the maximum median value (≈3 days) while, also in this case, *Errors* is the topic with the minimum median (≈10 hours). Again, the observed standard deviation is very large for each topic (minimum 2 months and a half, maximum ≈5 months), signifying that the $D(GHi_t)$ distributions are rather spread. As for Stack Overflow topics, we observed greater differences in values for the third quartile (minimum ≈2 days, maximum ≈2 weeks) with respect to the first quartile (minimum ≈30 minutes, maximum ≈4 hours), suggesting that the differences are more significant in the upper half of the distributions.

Similarly to Section IV-B, we employ the Friedman test to verify the presence of statistically significant differences across distributions. The result ($p-$value < 0.01) allows us to reject the null hypothesis that distributions of $D(GHi_t)$ across topics are not statistically significantly different. Analogously to Section IV-B, we use Nemenyi's test to perform all possible pairwise comparisons and order the topics based on their alive time. Also in this case we obtain a $p-$value < 0.01 in all comparisons for which we can reject the null hypothesis and the resulting ranking is displayed in Figure 4. We observe that the *Feature request* topic is the one that exhibits a longer median alive time. We hypothesize that the underlying reason is that this kind of topic is the only one that does not directly point to a bug, suggesting instead the addition or improvement of features, hence resulting in longer discussions. *Platform integration* follows, highlighting that indeed compatibility issues generally require a longer time to be addressed, hence significantly impacting development. The opposite instead can be stated for the topic *Errors*, which is the last in the ranking.

3) Stack Overflow and GitHub comparison: comparing the topics rankings among the two datasets, we observe the presence of the *Platform integration* topic among the top positions of both datasets, as it is the third most challenging topic to answer and the second in terms of time required to address related issues on GitHub. Instead, *Build & deploy* and *User interface*, the other two most challenging to answer topics, occupy a lower position in the GitHub ranking, evidencing that issues of this kind require a minor development effort to be fixed. Analogously, we observe that the topic *Errors* instead places at the bottom of both datasets' rankings.

• *Platform integration* is one of the most critical aspects, being the third most difficult topics to answer and the second in terms of time required to address related issues.

More in general, we observe that the topics obtained by applying LDA on GitHub issues deal with more diversified aspects and are more loosely related to application development. Nonetheless, we believe that GitHub can represent a useful source for the collection of additional data, as in the case of our study, or to investigate more varied aspects of the software development cycle. In addition, we observe a different order of magnitude in the distributions of $D(SOq_t)$ and $D(GHi_t)$ with the former showing a median closing time on the order of minutes and the latter on the order of days. We think this gap is not surprising, given that Stack Overflow posts enjoy much greater visibility than GitHub issues, but it suggests that, except for very specific project-related issues, developers should rely on the former to receive assistance more promptly.

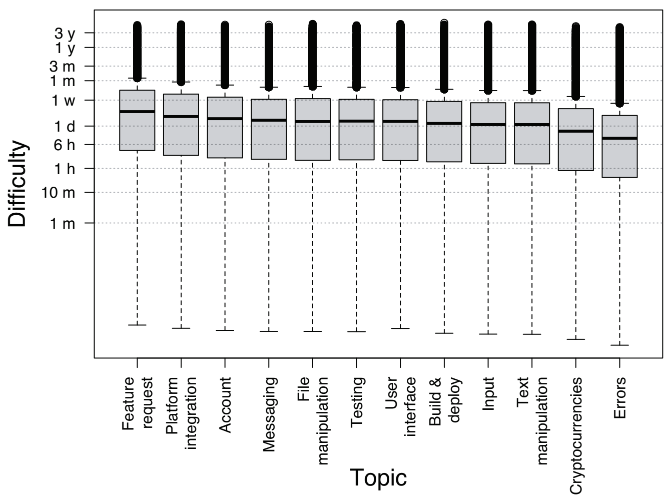


Fig. 4. Boxplot of $D(GHi_t)$ (difficulty of GitHub topics)

TABLE III
TOPICS IN THE GITHUB DATASET
(TOPICS IN ITALIC ARE IN COMMON WITH THE STACK OVERFLOW DATASET)

Topic	**Top words**	**Median** $D(GHi_t)$ **(minutes)**	$\sigma(D(GHi_t))$ **(minutes)**
Account	account user password login updat key chang creat	2,497.94	121,733.1
Build & deploy	instal packag build fail updat version linux releas	1,741.12	204,860.7
Cryptocurrencies	wallet sync connect mist ethereum contract eth ether	980.25	108,640.5
Errors	error uncaught read properti typeerror undefin enoent null	573.63	189,899
Feature request	request featur add support option suggestion disabl abil	4,236.85	150,930.4
File manipulation	file open folder save chang directori path drag	2,008.07	225,687.8
Input	shortcut search keyboard key tab select click input	1,622.91	196,381.5
Messaging	room messag user show invit list group chat	2,216.92	113,726
Platform integration	window icon start linux mac maco app crash	2,924.64	188,831.1
Testing	test error cypress run fail log browser chrome	2,083.38	147,853.8
Text manipulation	line code highlight text markdown render syntax charact	1,578.20	211,404.9
User interface	window bar theme scroll dark menu mode size	2,027.98	184,187

V. LIMITATIONS AND THREATS TO VALIDITY

In the following, we discuss the threats to the validity of our study according to the Cook and Campbell categorization [31].

Internal Validity: refers to the causality relationship between treatment and outcome [32]. We relied on Stack Overflow tags to identify posts related to desktop web apps development. As such, there is the possibility that some posts might have been missed during our posts selection, due to being mislabeled. To mitigate this threat, we performed the selection of the posts employing the significance and relevance measures, described in Section III-B1. Previous studies have found these measures to be effective in expanding the tags dataset and in limiting dataset noise [5], [33], [12], [10], [11], [14]. Another potential threat resides in the selection of the optimal number of topics K (14 for Stack Overflow, 13 in the case of GitHub), which potentially might have been sub-optimal, leading to the identification of topics that are too narrow or too general to extract meaningful insights from. We mitigated this threat by experimenting with different configurations and selecting the one that maximizes the generalizability and relevance of the topics based on the coherence measure [23]. Akin to the previous threat, the adopted procedure has been employed in previous studies that found it effective [5], [33].

Construct Validity: deals with the relation between theory and observation [32]. A potential threat comes from the labeling of the automatically generated topics, as the assigned names might not reflect the posts associated with the topics. We mitigate this threat by having the naming performed by one of the authors who possess experience in JavaScript development. During the naming procedure, in addition to relying on automatically produced top words, he manually sampled and analyzed a number of the most relevant documents for each topic. Assigned names were then reviewed and confirmed by a second author. Moreover, top words and examples for each topic are reported in the paper, to help the reader assess the relevance of assigned topic names. Additionally, we use the $D(SOq_t)$ and $D(GHi_t)$ metrics to measure the difficulty of identified topics, which might be a threat to construct validity, as these metrics might reflect more the (lack of) priority of a task rather than the difficulty of it. These metrics are a generalization of well-known ones used in other topic modeling studies [14], [10], [11], [12].

External Validity: deals with the generalizability of obtained results [32]. In our study, we mainly relied on data collected from Stack Overflow to identify the issues faced by developers of desktop web apps. However, this data might not be comprehensive of all the difficulties faced by developers, as there might be more subtle aspects that are rarely discussed. To mitigate this threat, we also investigated the prevalence of identified issues in real-world applications hosted on GitHub, identifying other aspects that are commonly discussed in them and evidencing that a number of previously identified issues are commonly discussed in GitHub issue reports. Moreover, in the analysis of GitHub issues, the adopted dataset consists mainly of Electron applications, with only a minority of NW.js apps. As such, there is the possibility that the obtained results are more pertinent to Electron itself, rather than to desktop web apps in general.

VI. Related work

In this section, we present the studies related to desktop web application development and discuss the work that applied topic modeling techniques to Stack Overflow data to elicit insights pertaining to the developer perspectives.

A. Desktop web apps

To the best of our knowledge, our previous work [4] is the only study, in the literature, that directly dealt with the topic of desktop web apps. In it, we conducted an investigation to characterize their usage and found preliminary evidence on some of the disadvantages associated with them. On a broader scope, the idea of employing the browser as a platform for the execution of cross-platform applications is not novel. In 2008, Taivalsaari and colleagues reported their experience in using a regular web browser as a platform for desktop applications [34]. In a subsequent work [35], the same authors, discuss the ever-narrowing boundary between the web and desktop applications. In the mobile domain, hybrid development frameworks allow developers to use web technologies for the development of their mobile applications. Malavolta et al. investigated the traits and the presence of hybrid mobile apps on the Google Play store [28]. In follow-up research, they focused on the differences perceived by end-users between hybrid and native mobile apps [36].

B. Topic modeling studies on Stack Overflow

There is a number of studies that applied topic modeling techniques to Stack Overflow data to extrapolate a variety of insights. Baruaet al. [37] were the first to investigate the general topics that the developer community discusses on Stack Overflow. In the mobile domain, Linares-Vasquez and colleagues [38] investigated the varied challenges that developers face when developing mobile applications while Rosen and Shihab [11] investigated the specific issues that arise on different mobile platforms. Other studies investigated needs and challenges of developers in multiple contexts: software security (Yang et al. [12]), machine learning (Alshangiti et al. [39] and Bangash et al. [40]), big data (Bagherzadeh et al. [10]), virtualization (Haque et al. [41]), blockchain (Wan et al. [33]), microservices (Bandeira et al. [42]), concurrency (Ahmed et al. [14]), usage of biometric APIs (Jin et al. [43]), and chatbot development (Adbellatif et al. [5]). More closely related to our work, Venkatesh et al. [44] investigate the challenges that web developers experience when using Web APIs and Bajaj et al. [45] investigated the common challenges faced by web developers. To the best of our knowledge, there is no study that explored the challenges faced by desktop web app developers on Stack Overflow. We believe that our study will be helpful to practitioners to understand the difficulties tied to the adoption of these technologies and to framework developers to improve them.

VII. Conclusions and Future work

We conducted an empirical study on 10,822 Stack Overflow posts related to the development of desktop web applications and issue reports of 453 open-source applications publicly available on GitHub. Results of our analysis evidence the presence of several issues related to the build and deployment processes for multiple platforms, reuse of existing libraries and tools, and interaction with native APIs.

As future work, we plan on investigating other aspects unexplored in this study that might represent other potential criticalities of desktop web app frameworks: being executed within a web browser, apps may suffer performance degradation and excessive energy consumption, especially if not properly optimized. Moreover, focusing on the GitHub dataset, we plan on analyzing the code of issue-fixing commits to understand and characterize how the problems evidenced in our study were solved.

References

[1] H.-B. Kittlaus and P. N. Clough, *Software product management and pricing: Key success factors for software organizations*. Springer Science & Business Media, 2008.

[2] I. Malavolta, "Beyond native apps: web technologies to the rescue!(keynote)," in *Proceedings of the 1st International Workshop on Mobile Development*, 2016, pp. 1–2.

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

[4] G. L. Scoccia and M. Autili, "Web frameworks for desktop apps: An exploratory study," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM'20. ACM, 2020.

[5] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in chatbot development: A study of stack overflow posts," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 174–185.

[6] R. Abdalkareem, E. Shihab, and J. Rilling, "What do developers use the crowd for? a study using stack overflow," *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.

[7] Stack Overflow, "Stack Overflow Developer Survey 2019," https://insights.stackoverflow.com/survey/2019, Accessed 07 January 2021.

[8] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[9] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A systematic mapping study of software development with github," *IEEE Access*, vol. 5, pp. 7173–7192, 2017.

[10] M. Bagherzadeh and R. Khatchadourian, "Going big: a large-scale study on what big data developers ask," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 432–442.

[11] C. Rosen and E. Shihab, "What are mobile developers asking about? a large scale study using stack overflow," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[12] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[13] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, A. Zaidman, Y. Kamei, and E. Hill, Eds. ACM, 2018, pp. 319–330. [Online]. Available: https://doi.org/10.1145/3196398.3196430

[14] S. Ahmed and M. Bagherzadeh, "What do concurrency developers ask about? a large-scale study using stack overflow," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[16] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "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, 2013, pp. 188–197.

[17] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 744–755.

[18] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 706–716.

[19] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[21] H. Jelodar, Y. Wang, C. Yuan, X. Feng, X. Jiang, Y. Li, and L. Zhao, "Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey," *Multimedia Tools and Applications*, vol. 78, no. 11, pp. 15 169–15 211, 2019.

[22] A. K. McCallum, "Mallet: A machine learning for language toolkit," *http://mallet. cs. umass. edu*, 2002.

[23] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proceedings of the eighth ACM international conference on Web search and data mining*, 2015, pp. 399–408.

[24] M. A. Stephens, "Edf statistics for goodness of fit and some comparisons," *Journal of the American statistical Association*, vol. 69, no. 347, pp. 730–737, 1974.

[25] W. Daniel, *Applied Nonparametric Statistics*, ser. Duxbury advanced series in statistics and decision sciences. PWS-KENT Pub., 1990. [Online]. Available: https://books.google.it/books?id=0hPvAAAAMAAJ

[26] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric statistical methods*. John Wiley & Sons, 2013, vol. 751.

[27] P. Nemenyi, "Distribution-free multiple comparisons," in *Biometrics*, vol. 18, no. 2. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962, p. 263.

[28] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "Hybrid mobile apps in the google play store: An exploratory investigation," in *2015 2nd ACM international conference on mobile software engineering and systems*. IEEE, 2015, pp. 56–59.

[29] dsanders11. Technical differences between electron and nw.js. [Online]. Available: https://github.com/electron/electron/blob/master/docs/development/electron-vs-nwjs.md

[30] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "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*, 2008, pp. 304–318.

[31] T. D. Cook, D. T. Campbell, and A. Day, *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston, 1979, vol. 351.

[32] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2012.

[33] Z. Wan, X. Xia, and A. E. Hassan, "What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities," *IEEE Transactions on Software Engineering*, 2019.

[34] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz, "Web browser as an application platform," in *2008 34th Euromicro Conference Software Engineering and Advanced Applications*. IEEE, 2008, pp. 293–302.

[35] T. Mikkonen and A. Taivalsaari, "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, 2011, pp. 22–26.

[36] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End users' perception of hybrid mobile apps in the google play store," in *2015 IEEE International Conference on Mobile Services*. IEEE, 2015, pp. 25–32.

[37] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[38] M. Linares-Vásquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using stack overflow," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 93–96.

[39] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? a study on stack overflow posts," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[40] A. A. Bangash, H. Sahar, S. Chowdhury, A. W. Wong, A. Hindle, and K. Ali, "What do developers know about machine learning: a study of ml discussions on stackoverflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 260–264.

[41] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in docker development: A large-scale study using stack overflow," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[42] A. Bandeira, C. A. Medeiros, M. Paixao, and P. H. Maia, "We need to talk about microservices: an analysis from the discussions on stackoverflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 255–259.

[43] Z. Jin, K. Y. Chee, and X. Xia, "What do developers discuss about biometric apis?" in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 348–352.

[44] P. K. Venkatesh, S. Wang, F. Zhang, Y. Zou, and A. E. Hassan, "What do client developers concern when using web apis? an empirical study on developer forums and stack overflow," in *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016, pp. 131–138.

[45] K. Bajaj, K. Pattabiraman, and A. Mesbah, “Mining questions asked by web developers,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 112–121.