Trends on Crowdsourcing JavaScript Small Tasks

Ioannis Zozas[a], Iason Anagnostou and Stamatia Bibi[b]

Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece

Keywords: JavaScript, Crowdsourcing, Small Tasks, Bountify.

Abstract: Crowdsourcing has become a popular alternative model for software development, aiming at assigning tasks to developers through an open call for participation. The major challenge when crowdsourcing software tasks, is to appropriately orient the tasks to ensure the participation of the community and increase the chances of getting a high-quality solution. Especially in constantly evolving development environments, such as JavaScript (JS) programming language and its applications, it is of high importance to be aware of the skills that can be acquired by the community, to successfully invest in crowdsourcing. In the current paper, we aim to explore trends when crowdsourcing small JS development tasks in an attempt to unveil a) the core technological skills that are more frequently required in the crowdsourced tasks, b) the functionalities that are more frequently crowdsourced, and c) the relationship between the technological skills and the functionalities crowdsourced. For this reason, we analysed 8-year contest data collected from Bountify crowdsourcing platform. The results showed that JS small task development does not focus on a single technology but on a series of technologies, frameworks and libraries that in most cases either overlap or complement each other.

1 INTRODUCTION

Crowdsourcing in Software Engineering focuses on overcoming task requirements acquisition by crowdsourcing them to stakeholders with a specific skill or domain knowledge (Wang, 2014). The crowdsourced software development model aims at recruiting stakeholders for software engineering tasks to reduce time-to-market, increase parallelism, lower cost and defect rates, and most importantly, alleviate the knowledge gap in a form of collective intelligence (Bibi, 2010) (Lakhani, 2010). This model enables extreme scalability for small tasks or self-contained microtasks, which is considered more reliable and efficient than a large task development scheme (LaToza, 2016). Industries can handle small tasks by crowdsourcing them to acquire missing knowledge and expertise, rather than outsourcing the whole development as one big task, which may risk the success of the project (Zanatta, 2018). As an emerging market, major software companies rely on crowdsourcing, boosting the popularity of crowdsourcing platforms with most notably Amazon Mechanical Turk, uTest, StackOverflow, TopCoder and Bountify. TopCoder has hosted more than 400 million competitions, uTest incorporated more than 100 thousand testers, and more than 16 million programming-related tasks have been answered in StackOverflow.

JavaScript (JS) on the other hand is today among the most popular programming languages for all-purpose development (Chatzimparmpas, 2019), presenting a constantly expanding development eco-system including front-end, client-side, server-side and Internet-of-things applications. It is a high-level dynamic, object-based, multi-paradigm, interpreted and weakly type language, which leverages a variety of libraries, frameworks and technologies. Large numbers of difficult to handle small tasks are based on JS due to the prementioned evolving and expanding ecosystem, as well as the fragmentation of required skills and technologies related. A common practice to deal with these small tasks is distributing them to an arbitrarily larger crowd via crowdsourcing (Guittard, 2015).

Under this scope, this study explores JS small task development, considering that the task is described in the form of a unique crowdsourced task that does not

[a] https://orcid.org/0000-0003-2159-1332

[b] https://orcid.org/0000-0003-4248-3752

belong to a network of micro-tasks on a project that is being crowdsourced. By analyzing the data derived from crowdsourced tasks our goal is to:

- Explore the trends related to a) the *technology* (i.e., frameworks) and the associated *programming languages* that these tasks use, and b) the *functionality* that the tasks implement and the *platforms* (i.e., operating systems, applications) on which these tasks are deployed.
- Associate *JS technologies to domain functionality, in an attempt to optimize small task completion efficiency*. In the case of the volatile JS development eco-system, trends can provide further information on the language current and future domain utilization (Mao, 2017) as well as reveal industrial requirement skills for developers to master (LaToza, 2016).

In the current study, we decided to shed focus on Bountify as a platform providing small tasks to the participants covering a variety of required programming languages, including JS. Bountify is a popular question-and-answer platform, that has been employed to improve crowdsourced small coding tasks, seeking to provide only code solutions and crowd-based support with an optional requirement of a payment or charity donation as a form or reward. We selected this platform among others as it currently hosts small crowdsourced task, it is open source, with free access, and easy access to the task database with tag-based filter. Bountify promoted crowdsourcing small task that lack the cognitive requirement to belong to a large project (LaToza, 2016).

The rest of the paper is structured as follows. Section 2 reviews the current related work regarding investigating trends in JS small tasks. Section 3 presents the case study design we utilized to locate and analyze trends. In Section 4 we present the results of the statistical analysis process while in Section 5 we discuss the results, implications to researchers and practitioners, as well as threads to validity. Finally, Section 6 concludes the paper.

2 RELATED WORK

Crowdsourcing software development is a research topic that concentrated the interest of the community the recent years. Yuen et. al. (2011) was the first to perform a survey on crowdsourcing development models and categorized them into four types; application, algorithm, performance and dataset. Hetmak (2013) conducted a systematic literature review in the domain of crowdsourcing systems to gain a better understanding of what crowdsourcing systems are and what typical design aspects are considered in the development of such systems. In addition, Alt et. al. (2010) described the concept of crowdsourcing by designing and implementing a crowdsourcing platform that integrates location as a parameter for distributing tasks to workers. In his concept, small tasks are broadcasted to a crowd in the form of open calls for solutions, concluding that as a concept is feasible. On the crowdsourcing task decomposition problem, Tong et. al. (2019) focused on how the desired reliability is achieved at a minimal cost. The authors proposed a series of efficient approximation algorithms using the greedy strategy and the optimal priority queue data structure to discover near-optimal solution based on cost and time (but not skills). The majority of papers focus on small tasks and most mainly on Amazon's Mechanical Turk micro-task market. Kittur et. al. (2008) examined the Amazon platform as a potential paradigm for engaging a large number of users for minimizing monetary costs and shortening time-to market. However, he pinpointed that further work is needed to understand the kinds of experiments that are well-suited to user testing via micro-task markets and determining effective techniques for promoting useful user participation and overall effectiveness. Moreover, Weidema et. al. (2016) concluded that many participants on the same platform find the crowdsourced tasks to be difficult despite the fact they are of limited scope.

With respect to the allocation of tasks to the crowd, Boutsis et. al. (2014) presented a crowdsourcing system that addresses the challenge of determining the most efficient allocation of tasks to the human crowd. This study concluded that crowdsourcing systems assign a varying task workload to a set of humans with different skills and characteristics. Furthermore, Bibi et al. (2010) analyzed among others the profiles, competencies and relative performance of the task participants with respect to the application domain of the crowdsourced task. On a different perspective, Machado et. al. (2017) conducted an empirical study to identify the difficulty in tally stakeholder programming skills and domain tasks. Kittur et. al. (2013) indicated the potential danger to replace some forms of skilled labor with unskilled labor in the task decomposition process. The authors concluded that task assignment in relation to each stakeholder's ability is an important issue for crowdsourcing labor economics, affecting efficiency, quality, overall cost and job satisfaction (Sun, 2017).

On the other hand, research on trends related to JS application development, is growing mainly due to

the popularity of the language. Gude et al. (2014) performed an empirical study on JS feature utilization and suggested possible future directions. Delcev et. al. (2018) performed a survey on frameworks and recorded trends in JS emerging web technologies. Sun et. al. (2017) and Rauschmayer (2012) explored both JS programming as well research trends. While research effort exists on JS trends, to our knowledge, there isn't currently any research on trends and common practices when it comes to crowdsourcing JS tasks.

3 CASE STUDY DESIGN

To empirically investigate and detect trends in crowdsourcing JS small tasks, we performed a case study on 771 JS coding tasks posted on Bountify platform between 2014-2021. The case study is performed according to the guidelines of Runeson et al. (2009).

3.1 Research Questions

The overall goal of the case study is formulated to the following three research questions:

- **RQ1:** Which are the *JS frameworks and technologies* mostly used by the crowdsourced tasks?

The purpose of this RQ is twofold: a) to identify trends regarding the development *frameworks* and the *technologies* (i.e., *programming languages)* when crowdsourcing JS tasks and b) to explore whether the *success* of JS crowdsourced tasks presents significant differences with respect to the technologies and programming languages used.

- **RQ2:** Which *types of functionalities* do the crowdsourced tasks mostly implement?

The purpose of this RQ is also twofold: a) to identify trends related to the *functionalities* that are mostly implemented by the crowdsourced tasks and the *platforms* on which the tasks will operate (i.e., operating systems, container applications-facebook, wordpress, etc.) b) to explore whether the *success* of JS crowdsourced tasks presents significant differences with respect to the functionalities required and the platform with which they are related to.

- **RQ3:** Is there a correlation between the *JS frameworks and technologies* used by the crowdsourced tasks and the *functionalities* implemented?

In the third research question, we investigate whether there is a significant correlation between *frameworks and technologies*, as well as *functionalities and*

platforms. Our goal is to identify trends in the technologies that are selected to implement particular types of functionalities and see whether the combination of technologies with functionalities can be successful when being crowdsourced.

3.2 Data Collection and Units of Analysis

For the purpose of our research, we collected data from tasks crowdsourced in the Bountify platform during the period from 2014 to March 2021. The collection process was performed by a data crawler developed by the second author.
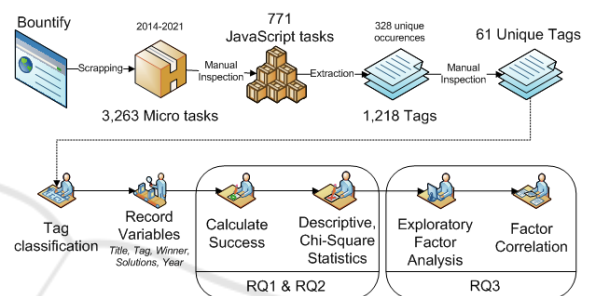

Figure 1: Overview of Data Collection and Analysis process.

In total, we collected data from 3,263 tasks. For each individual task, several variables were recorded:

- *Title*
- *Tag*
- *Winner*
- *Number of solutions*
- *Year of announcement*

In order to isolate tasks related to JS technology, we examined the tags and the titles of the tasks, and kept the tasks that included related keywords (i.e., js, javascript). The process was performed by two individual researchers of our team in order to ensure the accuracy of the results. This resulted in a total of 771 tasks.

To be able to perform the statistical analysis we quantified the absence or appearance of each tag in every task collected, by using a Boolean representation (0 or 1 correspondingly). A total number of 328 individual tags were recorded for all tasks. As tags were freely declared by each task provider, a significant number of similar tags were traced targeting the same (e.g., angular, angularjs) or different versions of software (e.g., bootstrap-3, bootstrap). Therefore, a manual grouping process was performed to summarize tags with the same meaning,

producing 61 individual tags counting 1,218 appearances in all tasks, as presented in Tables 2 and 3.

Additionally, for each task, we calculated the variable *task success* which is a Boolean variable that indicates whether at least one winning contribution exists, for the task under study. A successful posted task is considered a fulfilled task that has at least one winning contributor to it. For each unique tag then we calculated the **success** percentage that is described in Equation (1), where N_{solved} is the total occurrences of the particular tag in the tasks that acquire at least one acceptable solution, and N_{total} is the total tag occurrences of the particular tag in all contests analyzed.

$$Success_{\%} = N_{solved}/N_{total} \qquad (1)$$

The final step of the data collection process is to classify the tags into four groups: *frameworks* (JS frameworks)*, technologies (programming languages), functionalities* (i.e., visualization, data handling, security), and *platforms* (operating systems, container applications). An overview of the Data Collection and Analysis process is presented in Figure 1.

3.3 Data Analysis Methods

The applied methodology follows common statistical methods used by literature. To answer the first two RQ, we have calculated standard descriptive statistics, *observations,* and *frequencies*, and performed a *chi-square* test (IEEE, 2009) to determine whether each tag is associated with *success* as described by equation (1). To answer the third RQ, we performed Exploratory Factor Analysis (EFA) (Snook, 1989), to determine the underlying dimensionality of tags based on the correlation among them. EFA is a statistical approach used to examine the internal reliability of a measure and investigate the theoretical constructs, or factors, that might be represented by a set of items. For this purpose, we performed Principal Component Analysis (PCA) to decide the total number of factors. The approach is used to discover the factor structure of a measure and to examine its internal reliability, usually when no hypotheses exist about the nature of the underlying factor structure of their measure (Difallah, 2015). In order to perform EFA and determine components of related tags in the context of JS crowdsourced development, we followed the steps described in (Papoutsoglou, 2017). We performed PCA factoring extraction with the adaption of the Varimax rotation of the data. In order to determine the number of factors, we selected all factors with an eigenvalue

higher than 1. The commonality of each variable was examined with a cutoff value of 0.5, whereas in the cases where the cut-off values were lower than 0.5 the model was refitted. Additionally, for RQ3 we performed Spearman correlation in order to explore potential correlations between the derived *frameworks and technologies* factors, and the *functionality and platform* factors.

4 RESULTS

In the current section, we present the results of the case study performed to answer the three research questions.

4.1 RQ1

In this RQ we want to explore which JS frameworks and technologies (i.e., programming languages related to JS) are mostly used by crowdsourced small tasks and whether these tasks are successful. In order to answer this RQ, we analyzed the tags that are used by the contest providers to describe each crowdsourced task and define the skills required to address it. Table 1 presents the frequency, the percentage frequency, and the success percentage for each tag identified along with the results of the chi-squared test performed for each tag and the *success* variable.

In the case of **frameworks**, we identified a large diversity of both client and server-side technologies. The dominant development framework is *"jQuery"* followed by *"bootstrap"*. When analyzing the frameworks tag frequencies, we observed that there is a variety of frameworks used just one time (i.e., Parsley.js, Popper.js, Chart.js, etc.) which were all merged in *"jslibrary"* tag. However, the combined frequency of all these frameworks is high, indicating the wide diversity in framework usage. The frequency of each individual library is mere below 2 instances for each one. A future further analysis with a larger dataset should indicate whether these libraries should be merged or not for further identification. To summarize with framework usage, special emphasis should be placed upon *"nodejs"* which seems to be very popular probably due to the fact that it supports server-side scripting.

When it comes to **programming languages,** we can see that *"css"* and *"html"* scripting languages are often used complementary within JS applications. This finding can be explained by the fact that many tasks are not implemented in any framework. Most frameworks are oriented to directly manipulate the

output and as a consequence, the absence of a framework requires, most of the time, the existence of code implemented in the relevant output formatting languages (CSS, HTML). Regarding JS interoperability with other full-stack development programming languages, *"php"* is the most prominent, followed by "ruby" and *"python"*.

Table 1: JS Frameworks & technologies tags.

Framework	N	%	Success	χ^2	p .05
jquery	182	23.61	96.70	1.93	0.16
bootstrap	79	10.25	94.94	0.01	0.91
jslibrary	61	7.91	93.44	0.20	0.65
nodejs	42	5.45	83.33	11.36	**0.01**
angular	20	2.59	85.00	3.82	0.05
react	18	2.33	83.33	4.71	**0.03**
meteor	12	1.56	91.67	0.22	0.63
typescript	9	1.17	77.77	5.16	**0.02**
vue	7	0.91	100	0.39	0.52

Languages	N	%	Success	χ^2	p .05
css	123	15.95	93.50	0.40	0.52
html	108	14.01	94.44	0.01	0.90
php	37	4.80	86.49	5.18	**0.02**
json	27	3.50	92.59	0.24	0.62
ajax	14	1.82	92.86	0.09	0.75
xml	9	1.17	100	0.51	0.47
ruby	8	1.04	77.78	6.22	**0.01**
python	7	0.91	100	0.39	0.52
java	5	0.65	60.00	12.02	**0.01**
perl	3	0.39	100	0.16	0.68
c	2	0.26	100	0.11	0.73

Finally, concerning **data manipulation**, *"json"* and *"xml"* are the most common technologies used for data retrieval. Figure 2 presents a relative representation of the 10 most frequently appearing framework and technology tags from 2012 to 2020. We can notice that the use of *React* is increasing while all others decrease (especially in the dominant case of *jQuery*).
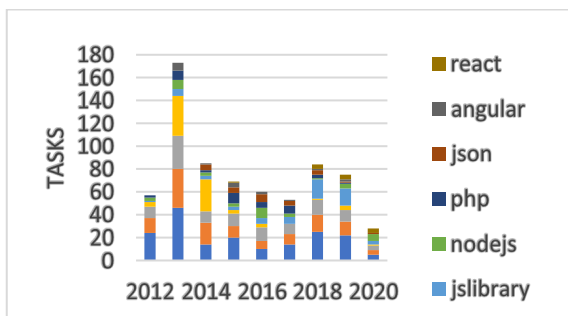

Figure 2: Analysis of small tasks framework utilization.

Regarding the success rate we can observe that tasks related to *"typescript"*, *"jquery"* and *"bootstrap"* frameworks present high percentages of success, followed by tasks implemented in *"nodejs"*, *"react"* and *"angular.js"*. We can observe that the least successful tasks are related to *"ruby"* and *"python"* programming languages and *"vue.js"* framework. In order to examine whether the tasks implemented in the various frameworks and technologies present a significant difference with respect to *success*, we conducted a chi-square test for each related tag and the success variable. For most tasks, we calculate a χ^2 value with an asymptotic 2-sided significance p-value over 0.05. The only frameworks that present a significant difference with respect to *success* are *"nodejs"*, *"react"* and *"typescript"*, while for languages *"php"*, *"ruby"* and *"java"*. This leads to the conclusion that regarding RQ1, there are no significant differences in success percentages of the tasks regardless of the frameworks and programming languages implemented.

4.2 RQ2

In this RQ we will first identify trends related to the types of applications and the functionalities that are crowdsourced in JS tasks and then explore whether the success of the tasks presents significant differences within each type of functionality, an application implemented, and platform. In Table 2 we present the descriptive statistics of the tags along with the results of the chi-square tests. In Figure 3 we present the 10 most frequent platform and functionality tags from 2012 to 2020.
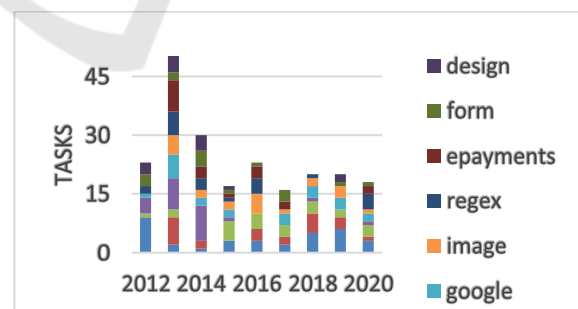

Figure 3: Comparison of top 10 tag frequency of each category.

In the case of *platforms and applications*, *"wordpress"* is the dominant content management platform confirming the largest market share to other content management systems (*"cms"*) observed, while *"google"* is the dominant service provider. We observe that the demand concerning *operating systems* (such as *"windows"*, *"linux"*, *"android"* and *"ios"*) is very limited. Probably this can be explained

by the shift towards the cloud computing paradigm where Software-as-a-Service operational models override the client operating system (Sharma, 2011). On the other hand, in the case of *core functionality*, the dominant functionality is visual representation (*"layout"*, *"canvas"*, *"image"*, *"design"*, *"table"*).

Table 2: Tags analysis.

Platform & Application	N	%	Success	χ^2	p .05
wordpress	24	3.11	91.7	0.44	0.50
google	23	2.98	91.3	0.53	0.46
browser	14	1.82	92.9	0.09	0.75
cms	11	1.43	81.8	3.66	0.05
mobile	10	1.30	90.0	0.44	0.50
android	7	0.91	85.7	1.12	0.28
evnote	7	0.91	100	0.39	0.52
facebook	6	0.78	100	0.34	0.56
ios	6	0.78	83.3	1.54	0.21
excel	6	0.78	100	0.34	0.56
youtube	5	0.65	100	0.28	0.59
linux	5	0.65	80.0	2.15	0.14
web	5	0.65	80.0	2.15	0.14
videogame	4	0.52	100	0.22	0.63
pdf	3	0.39	100	0.16	0.68
windows	3	0.39	66.6	4.69	**0.03**
twitter	1	0.13	100	0.05	0.81

Core functionality	N	%	Success	χ^2	p .05
layout	34	4.41	100	1.99	0.15
canvas	26	3.37	96.2	0.11	0.73
database	24	3.11	83.3	6.33	**0.01**
image	21	2.72	90.5	0.75	0.38
epayments	19	2.46	94.7	0.00	0.99
form	16	2.08	93.8	0.02	0.86
design	15	1.95	93.3	0.05	0.81
table	14	1.82	100	0.80	0.37
algorithm	18	2.33	83.3	4.71	**0.03**
regex	21	2.72	100	1.21	0.27
video	12	1.56	100	0.68	0.08
security	10	1.30	90.0	0.44	0.50
analytics	9	1.17	100	0.51	0.47
geolocation	8	1.04	75.00	6.22	**0.01**
responsive	8	1.04	100	0.45	0.50
sales	8	1.04	87.5	0.82	0.36
api	7	0.91	100	0.39	0.52
audio	7	0.91	100	0.39	0.52
messaging	7	0.91	71.4	7.58	**0.00**
datatransfer	6	0.78	100	0.34	0.56
graphics	6	0.78	100	0.34	0.56
performance	5	0.65	100	0.28	0.59
mail	4	0.52	100	0.22	0.63

Although there are JS frameworks that can easily support this functionality, it seems that the absence of a framework, in most contests, contributes to these high frequencies that are reported. A significant functionality group is *logic-related*, including *"algorithms"* and regular expressions (*"regex"*) that support web application development. Lastly, *"epayments"* and *"sales"* also present high frequencies, indicating that JS client-side technologies are used to minimize data transitions to servers and ensure data safety and privacy. Regarding the demand for specific functionalities and applications over years, as presented in Figure 3 we can see that this does not remain stable. The popular tags tend to present a declining trend while new tags appear over years that concentrate the attention (i.e *"responsive"*).

4.3 RQ3

Our purpose in this RQ is to explore whether we can form components of related tags with respect to a) the different technology, platform and framework used b) the different functionality and application implemented. In the end, our target is to examine if these components are correlated.

Table 3: Frameworks and Technologies factor analysis.

Component	Initial Eigenvalues			Success %	Tags
	Total	% Variance	Cumulative %		
1	1.76	8.39	8.39	100	python, perl, c
2	1.60	7.62	16.01	94.3	xml, json
3	1.48	7.06	23.08	93.3	html, css, java
4	1.42	6.77	29.86	85.4	nodejs, meteor
5	1.29	6.14	36.00	87.0	react, typescript
6	1.19	5.69	41.70	89.4	perl, c, php, ajax
7	1.18	5.65	47.35	95.6	bootstrap, jquery, angular, java
8	1.06	5.08	52.44	87.8	c,ajax, ruby, angular
9	1.02	4.89	57.33	91.5	jslibrary, java, vue

The components of *JS frameworks and technologies* and the related *success* percentages for each component are presented in Table 3. The success percentage is derived according to Equation (1), and as each task includes multiple tags, for each tag, success is based on the success task in which it is included. The calculation of each tag success is based on the mean of cumulative success rates of each task containing at least the corresponding tag. The tag

inclusion in each task is based on the process described in Section 3.2.

The components are presented in descending order of importance based on the percentage of the explained variance. The cumulative coverage for frameworks and technologies consists of 9 components explaining 57.33% of the total variance. The Kaiser-Meyer-Olkin (KMO) measure of sampling adequacy was 0.521. All sampling adequacies were above the commonly recommended value of 0.5, while the Bartlett's test of sphericity was statistically significant $x^2(210) = 1,046.96$ for $p < 0.001$. We also note that all eigenvalues score high on success over 87%. Factors revealed distinguishable and highly related frameworks and technologies found in tasks.

Table 4: Functionalities factor analysis.

Component	Initial Eigenvalues			Success %	Tags
	Total	% Variance	Cumulative %		
1	2.44	6.11	6.11	93.3	table, browser
2	1.88	4.70	10.81	94.1	windows, linux, android, ios
3	1.62	4.05	14.86	93.8	design,responsive
4	1.47	3.68	18.54	91.3	android,datatransfer mobile, ios
5	1.41	3.52	22.07	98.0	layout, excel, form
6	1.37	3.44	25.52	89.5	sales, database, security
7	1.32	3.31	28.83	92.6	google, api
8	1.30	3.25	32.09	87.5	analytics, algorithm
9	1.23	3.09	35.18	100	api, performance, graphics
10	1.21	3.03	38.22	100	facebook, mail
11	1.20	3.01	41.23	94.7	tweeter, epayments
12	1.15	2.88	44.12	100	videogame,graphics
13	1.14	2.85	46.97	96.1	layout, image, youtube, canvas
14	1.12	2.82	49.79	90.3	mobile, ios,security, evnote
15	1.08	2.70	52.50	82.4	messaging, cms
16	1.05	2.64	55.14	85.7	web, geolocation
17	1.02	2.56	57.70	95.6	wordpress, regex
18	1.01	2.52	60.23	100	video
19	1.00	2.51	62.74	100	regex, audio
20	1.00	2.50	65.25	100	pdf

In Table 4 we present the results of PCA analysis performed to identify factors based on the types of functionalities and applications implemented. In total 20 components are identified explaining 65.26% of variance. Each eigenvalue consists of at least one tag to a maximum of four. Concerning Functionalities,

the KMO measure of sampling adequacy was 0.527. All sampling adequacies were above the recommended value of 0.5, while Bartlett's test of sphericity was statistically significant $x^2(780) = 3,499.98$ for $p < 0.001$. We note that all eigenvalues score high on success over 82%.

Table 5: Tag component correlation analysis.

JS Frameworks and Technologies		Functionalities		rho	Success
C1	Tags	C2	Tags		
9	jslibrary, java, vue	13	layout, image, youtube, canvas	.287	95.0
6	perl, c, php, ajax	6	sales, database, security	.241	90.4
4	nodejs, meteor	6	sales, database, security	.214	86.6
5	react, typescript	8	analytics, algorithm	.214	86.0
4	nodejs, meteor	2	windows, linux, android, ios	.187	87.9
5	react, typescript	1	table, browser	.187	91.1
1	python, perl, c	20	pdf	.186	100
2	xml, json, evernote	14	mobile, ios, security, evernote	.179	91.5
1	python, perl, c	19	regex, audio	.173	100
1	python, perl, c	2	windows, linux, android, ios	.152	95.6
4	nodejs, meteor	14	mobile, ios, security, evernote	.138	87.5
5	react, typescript	13	layout, image, youtube, canvas	.138	94.3
1	python, perl, c	17	wordpress, regex	.128	96.0
1	python, perl, c	4	android, mobile, datatransfer, ios	.124	93.3
7	bootstrap, java, jquery, angular	19	regex, audio	-.113	95.9
4	nodejs, meteor	4	android, mobile, datatransfer, ios	.112	87.8
5	react, typescript	4	android, mobile, datatransfer, ios	.112	88.1
4	nodejs, meteor	15	messaging, cms	.107	86.8
5	react, typescript	14	mobile, ios, security, evernote	.107	87.7
3	html, css, java	19	regex, audio	-.106	94.1
8	c, ajax, ruby, angular	6	sales, database, security	.105	89.0
1	python, perl, c	14	mobile, ios, security, evernote	.100	92.1
6	perl, c, php, ajax	7	google, api	.099	89.8
7	bootstrap, java, jquery, angular	14	mobile, ios, security, evernote	-.096	94.9

[a.] $P < 0.001$, C for Component

As a next step, we performed correlation analysis to identify significant correlations between framework and technology components, and, functionality and application components. The analysis indicated statistical significance (Spearman's rho coefficient for

p < 0.001) as presented in Table 5 revealing twenty statistically significant correlations.

From the examination of the statistically significant correlations, we can derive some interesting findings:

- C1.1 component (*"python"*, *"perl"*, *"c"*) is the most correlated component from the framework and technology components, as it is associated with 6 functionality and application components. It seems that these three programming languages are used supplementary with JS language to perform a variety of functionalities related to operating systems (C2.1, C2.2, C2.14), regular expressions (C2.17) and mobile data transferring and security handling (C2.4, C2.14).

- C1.4 (*"nodejs"*, *"meteor"*) and C1.5 (*"react, "typescript"*) are both correlated to mobile data transfer functionalities (C2.4). C1.4 is also related to back-office operations (C2.6) and content management (C2.15). C1.5 is also related to user interface functionalities (C2.1, C2.13) and to more demanding tasks related to analytics and algorithms (C2.8).

- C1.6 component (*"perl"*, *"c"*, *"php"*, *"ajax"*) correlates to back-office functionalities (C2.6) and functionalities related to the interaction with external sources (C2.7). C1.7 (*"bootstrap"*, *"java"*, *"jquery"*, *"angular"*) correlates to functionalities related to regular expressions.

- We should note that all correlations are over 84% *successful*, meaning that the contests that are related to the technologies and functionalities that are present in the components have managed to acquire a winning solution. Specifically component C1.1 (*"perl"*, *"python"*, *"c"*) when combined with component C2.17 (*"regex"*, *"audio"*) or component C2.20 (*"pdf"*) it presents 100% *success*.

5 DISCUSSIO1N

5.1 Interpretations of Results

Based on the above results in Section 4.1, we can identify several trends when crowdsourcing JS micro-tasks. With respect to JS *frameworks and technologies*, *Node.JS* and *Meteor* technologies are widely used for crowdsourcing a variety of tasks, including *business* and *data-oriented* tasks for several platforms. These frameworks are also found when crowdsourcing *mobile deployment* and *security-related* tasks. On the other hand, *React* and *TypeScript* are also used in *multiple platform*

development, but emphasize more in *layout and user interface* micro-tasks. A number of programming languages have been often utilized supplementary with JS for development. These include *Python*, *Perl* and *C* languages. In these cases, these languages are used to handle tasks related to *operating systems* and for *regular expression manipulation*.

In Section 4.2 with respect to the JS *functionalities and applications* that are mostly crowdsourced, we observe that tasks related to *mobile applications* and *security* are common and they are implemented by a disperse variety of technologies. These may include other programming languages like *Python* or JS frameworks like *React* and *Bootstrap*, as well as technologies to distribute data as *XML* and *JSON*. *Sales transactions, database manipulation* and *security tasks* rely on server-side technologies like *Node.JS* and *Meteor*, as well as on other programming languages in conjunction with JS like *C*, *PHP*, and *AJAX*. In these cases, we assume that *AJAX* calls act as an intermediate to ensure the interoperability of these technologies. We should mention that the above technologies also pose a trend in mobile data operations and mobile platforms, most notably *Android*. An interesting output is that concerning the mobile platform development, for *Android*, most preferred JS technologies are *Node.JS*, *Meteor*, *React*, and *TypeScript*, while for *iOS*, above the prementioned, *Bootstrap*, *jQuery*, and *Angular*.

In Section 4.3 we conclude that the expanded eco-system of JS crowdsourced micro-tasks is not focused on a single technology but rather on a series of technologies, frameworks, and libraries that in most cases either overlap or complement each other. Also, we can say that JS micro-tasks are related to the implementation of a variety of functionalities the most successful of which are *user-interface handling*, *layout tasks*, and *regular expression manipulation*. All these tasks can be considered small light-weight tasks. Tasks that can be considered more demanding, in the sense that they require more specialized skills, such as tasks related to *operating systems*, *business logic*, and *algorithms* tend to present lower success percentages, but still above 80% in most of the cases.

5.2 Implications to Researchers and Practitioners

The results of the current study can be used both by researchers and practitioners. The current findings indicate that crowdsourcing JS small tasks can be particularly successful since most contests are able to acquire a winning solution. This is an indication that *researchers* can work on models for decomposing

larger JS applications into a series of small tasks that can be successfully and easily crowdsourced. Additionally, we can observe that particular JS functionality such as regular expression manipulation, video, and audio handling has been crowdsourced with a variety of programming languages and frameworks as presented in Table 5. This is a sign that there is a need to create libraries or frameworks that can handle such functionalities. Furthermore, while crowdsourcing has been a popular research topic, researching crowdsourcing small tasks has been mostly based on the Amazon Mechanical Turk platform (Kittur, 2008). Future comparison of different small tasks crowdsourcing platforms, tasks, and contributors could further reveal trends in JS technologies and functionalities. Regarding *Practitioners*, the current findings on trends in JS development indicate indirectly demands for skills and domain professionality in the job market. By recording the technologies and the related functionalities that are mostly crowdsourced in JS micro-tasks practitioners can benefit by acquiring the related skills to be able to correspond to present and future industry demands.

5.3 Threats to Validity

Based on the categorization of Runeson et. al. (2009), we will discuss the threats to validity identified in this study. Regarding *Construct Validity*, we should mention that the current metrics as described (and currently the tags of each small task) may affect the outcome of the overall findings. We cannot deny that the evaluation of alternative metrics that have not been participated in our study should not be included in the future. Such metrics may include developer experience, skills, and trends in JS development technologies (Meldrum, 2017). Regarding *Internal Validity*, our study attempts to detect trends. However, we do not claim that the presented results are from any form of causality rather than trends. Regarding *Reliability*, the process followed in our study has been thoroughly documented in the Case Study Design section in order to be easily reproduced. Thus, we believe that the replication of our study is safe and the reliability is ensured. Regarding *External Validity*, changes in the findings might occur in cases of either small task dataset alternations or the use of different small task platforms. Future replication of the current study would be valuable to verify our findings and support the generalizability supposition.

6 CONCLUSIONS

In the current study, we explored trends in crowdsourcing small tasks developed in JS. Our aim is to associate domain functionality and JS-related technology and frameworks. In total we have analysed 771 JS small tasks, crowdsourced in the Bountify platform. The results show that the JS crowdsourced tasks are successful in their majority and that a series of frameworks (jQuery, Bootstrap, Node.JS) and languages (HTML, CSS, XML) are employed for implementing tasks related to visualization (user interfaces, layout), data manipulation (security, databases, algorithms) and platform deployment (iOS, Windows, Android). Overall, we conclude that the expanded eco-system of JS crowdsourced micro-tasks is not focused on a single technology but rather on a series of technologies, frameworks, and libraries that in most cases either overlap or complement each other.

REFERENCES

Alt, F., Shirazi, A., Schmidt, A., Kramer, U., Nawaz, Z. (2010). Location-based crowdsourcing: extending crowdsourcing to the real world, *6th Nordic Conf. on Human-Computer Interaction*, Association for Computing Machinery, NY, USA, 13–22.

Archak, N. (2010). Money, glory and cheap talk: Analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on TopCoder.com, *19th International Conference on WWW*, NY, USA, 21–30.

Bibi, S., Zozas, I., Ampatzoglou, A., Sarigiannidis, P. G., Kalampokis, G., Stamelos, I. (2020). Crowdsourcing in Software Development, *IEEE Access*, vol. 8, 58094-58117.

Boutsis, I., Kalogeraki, V. (2014). On Task Assignment for Real-Time Reliable Crowdsourcing, *IEEE 34th Int. Conf. on Distributed Comp. Systems*, Spain, 1-10.

Chatzimparmpas, A., Bibi, S., Zozas, I., Kerren, A. (2019). Analyzing the Evolution of Javascript Applications. *14th International Conference on Evaluation of Novel Approaches to Software Engineering*, vol. 1, 359-366.

Delcev, S., Draskovic, D. (2018). Modern JavaScript frameworks: A Survey Study, *Zooming Innovation in Consumer Technologies Conference*, Serbia, 106-109.

Difallah, D., Catasta, M., Demartini, G., Ipeirotis, P., Cudré-Mauroux, P. (2015). The Dynamics of Micro-Task Crowdsourcing: The Case of Amazon MTurk, *24th Int. Conf. on WWW,* Switzerland, 238–247.

Gude, S., Hafiz, M., Wirfs-Brock, A. (2014). JavaScript: The Used Parts, *IEEE 38th Annual Computer Software and Applications Conference*, Sweden, 466-475

Guittard C., Schenk E., Burger-Helmchen T. (2015). Crowdsourcing and the Evolution of a Business Ecosystem. *Advances in Crowdsourcing*. Springer.

Hetmank, L. (2013). Components and Functions of Crowdsourcing Systems – A Systematic Literature Review. *Wirtschaftsinformatik Proceedings*, 4.

1061-1998: IEEE Standard for a Software Quality Metrics Methodology, *IEEE Standards, IEEE Computer Society*, 31 December 1998 (reaf. 9 December 2009).

Kittur, A., Chi, E., Suh, B. (2008). Crowdsourcing user studies with Mechanical Turk. *Conference on Human Factors in Computing Systems*, NY, USA, 453–456.

Kittur, A., Nickerson, J., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., Lease, M., Horton, J. (2013). The future of crowd work. *Conference on Computer supported cooperative work*. Association for Computing Machinery, NY, USA, 1301–1318.

Lakhani, K., Garvin, D., Lonstein, E. (2010). TopCoder(A): Developing software through crowdsourcing. Harvard Business School Case

LaToza, T., Van der Hoek, A. (2016). Crowdsourcing in Software Engineering: Models, Motivations, and Challenges, *IEEE Software*, vol. 33, 1, 74-80, Jan.-Feb.

Machado, M., Zanatta, A., Marczack, S., Prikladnicki, R. (2017). The Good, the Bad and the Ugly: An Onboard Journey in Software Crowdsourcing Competitive Model, *4th Int. Workshop on Crowd Sourcing in Soft. Engin.*, Buenos Aires, Argentina, 2-8.

Mao, K., Capra, L., Harman, M., Jia, Y. (2017). A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software*, Vol. 126, 57-84

Meldrum, S., Licorish, S., Savarimuthu, B. (2017). Crowdsourced Knowledge on Stack Overflow. *21st Int. Conf. on Evaluation and Assessment in Software Engineering*. Association for Computing Machinery, NY, USA, 180–185.

Papoutsoglou, M., Mittas, N., Angelis, L. (2017). Mining People Analytics from StackOverflow Job Advertisements, *43rd Euromicro Conf. on Soft. Eng. and Advanced Applications*, Austria, 108-115.

Rauschmayer, A. (2012). The Past, Present, and Future of JavaScript. O'Reilly Media, Inc.

Runeson, P., Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empir Software Eng* 14, 131.

Sharma R., Sood M. (2011). Cloud SaaS: Models and Transformation. *Advances in Digital Image Processing and Information Technology. Communications in Computer and Information Science*, vol 205. Springer.

Snook, S., Gorsuch, R. (1989). Principal component analysis versus common factor analysis: A Monte Carlo study. *Psychological Bulletin*, 106, 148-154.

Sun, K., Ryu, S. (2017). Analysis of JavaScript Programs: Challenges and Research Trends. *ACM Comput. Surv*. 50, 4, Article 59 (November 2017), 34.

Tong, Y., Chen, L., Zhou, Z., Jagadish, H.V., Shou, L., Lv, W. (2019). SLADE: A smart large-scale task decomposer in crowdsourcing. *35th IEEE Int. Conf. on Data Engin.*, ICDE Macau, China. 2133-2134

Wang, H., Wang, Y., Wang, J. (2014). A participant recruitment framework for crowdsourcing-based software requirement acquisition, *9th IEEE Int. Conference on Global Software Engineering*, 65–73.

Weidema, E., López, C., Nayebaziz, S., Spanghero, G., Van der Hoek, A. (2016). Toward microtask crowdsourcing software design work, *3rd Int. Workshop on Crowd Sourcing in Soft. Engin.*, NY, USA, 2016, 41–44.

Yuen, M., King, I., Leung, K. (2011). A Survey of Crowdsourcing Systems, *3rd Int. Conf. on Privacy, Security, Risk & Trust*, MA, USA, 766-773.

Zanatta, A., Machado, L., Steinmacher, I. (2018). Competence, Collaboration, and Time Management: Barriers and Recommendations for Crowdworkers, *5th Int. Workshop on Crowd Sourcing in Soft. Engin.* (CSI-SE), Gothenburg, Sweden, 9-16.