Using Docker to Assist Q&A Forum Users

Luis Melo, Igor Wiese and Marcelo d'Amorim

Abstract—Q&A forums are today a valuable tool to assist developers in programming tasks. Unfortunately, contributions to these forums are often unclear and incomplete. Docker is a container solution that enables software developers to encapsulate an operating environment and could help address reproducibility issues. This paper reports on a feasibility study to evaluate if Docker can help improve reproducibility in Stack Overflow. We started surveying Stack Overflow users to understand their perceptions on the proposal of using Docker to reproduce Stack Overflow posts. Participants were critical and mentioned two important aspects: cost and need. To validate their criticism, we conducted an exploratory study focused on understanding how costly the task of creating containers for posts is for developers. Overall, results indicate that the cost of creating containers is not high, especially due to the fact that dockerfiles are highly similar and small. Based on these findings we developed a tool, dubbed FRISK, to assist developers in creating containers for those posts. We then conducted a user study to evaluate interest of Stack Overflow developers on the tool. We found that, on average, users spent nearly ten minutes interacting with FRISK and that 45.3% of the 563 FRISK sessions we created for existing posts resulted in a successful access to the corresponding web service by the owners of the post. Overall, this paper provides early evidence that the use of Docker in Q&A forums should be encouraged for configuration-related posts.

1 INTRODUCTION

Question and Answer (Q&A) forums, such as Stack Overflow, has become widely popular among software developers. Unfortunately, it is not uncommon to find posts in Q&A forums with problematic instructions on how to reproduce issues [1]–[4]. For example, Mondal et al. [4] recently found that 68% of the code snippets they analyzed in Stack Overflow posts required minor and major modifications. These findings indicate that *addressing reproducibility issues* is important to improve user's experience in Q&A forums.

This paper evaluates the extent to which container technology can mitigate the reproducibility problem. Containers enable developers to encapsulate operating environments, a key aspect for reproduction. Docker is the main representative of that technology today, holding 83% of the market share [5] and supporting Continuous Deployment in software projects [6], [7]. Conceptually, forum users can document their issues and fixes more objectively with containers.

The paper reports on a study to assess the feasibility of using Docker [8] to address reproducibility issues in Stack Overflow. The study is organized in three complementary parts: 1) Survey, 2) Exploratory Study, and 3) User Study. The survey is essential to evaluate the perceptions of developers about the use of Docker in Stack Overflow. Different factors can affect developers' opinion on the idea, including inexperience with Docker, simplicity of posts, and concerns with security. The exploratory study is important to validate the opinion of users during the survey. For example, if posts are either too complex to reproduce or too simple to reproduce, developers may feel discouraged to add another layer of complexity to the posts. The exploratory study was designed to answer the main concerns developers raised during the survey. Finally, the user study is important to evaluate, in a more practical setting, how developers perform when using a tool we created to facilitate the creation of containers. In the following, we briefly detail each part of our study.

Survey. The starting step of our feasibility study was to run a survey to understand the opinion of Stack Overflow users about our proposal. The main observations we made from the survey were that 1) developers already have a good understanding of Docker (e.g., 35.5% of the participants use it frequently) and that 2) the main concerns of developers about our proposal are related to the human cost of writing Dockerfiles and the need to write them.

Exploratory Study. To evaluate the plausibility of the concerns raised by developers during the survey, we ran an exploratory study where we analyzed their main concerns. We analyzed these concerns under the light of 600 Stack Overflow posts we tried to reproduce with Docker. The key concerns were need, cost, and realizability. Considering the aspect need, indeed, we found that a straight answer typically suffices to address general posts, such as posts asking how to use a given API. Reproducibility becomes a more important issue for configuration-related posts, which are also common in this domain and whose questions and answers are more involved. To sum up, we found that none of the concerns developers raised was a showstopper. As such, we decided to run a user study to monitor user activity.

User Study. The exploratory study showed that productivity in creating containers could be improved if a system existed to support the creation of similar containers. That motivated us to create FRISK. To start a reproduction session in FRISK, a user needs to select an icon card, from a card deck, indicating his choice of container configuration. The card deck includes the most common configurations we found in Stack Overflow for server-side web development, which is the domain we focused on this study. Sessions created in FRISK are stored in the cloud, and a link for those sessions can be shared in Stack Overflow. FRISK is an online integration tool for Docker and Q&A forums; Stack Overflow

• *L. Melo is (currently) with C.E.S.A.R., Brazil. E-mail: lhsm@cesar.org.br. Igor Wiese is with the Department of Computing, Federal University of Technology - Paraná (UTFPR), Brazil E-mail: igor@utfpr.edu.br M. d'Amorim is with the Center of Informatics, Universidade Federal de Pernambuco, Brazil. E-mail: damorim@cin.ufpe.br.*

is a case in point. The goal of the user study we conducted with FRISK is to evaluate interest of real developers on a tool that could be used to integrate Stack Overflow and Docker. Over the course of a month, we monitored user activity of a total of 563 FRISK sessions we created for a total of 200 Stack Overflow posts created by the developers we monitored. We found that users spent nearly ten minutes, on average, interacting with FRISK and that 255 of the 563 (=45.3%) sessions resulted in a successful access to the web service spawned with the execution of the container, i.e., users were able to build the image, run the container, and access the corresponding service from a web browser. We interpreted this result as a positive indication of the potential of the tool to assist Q&A forum users.

In summary, our results suggest that linking Docker containers to Q&A forums is an interesting alternative to improve reproducibility of configuration posts. The artifacts produced in this study are publicly available [9] and FRISK is publicly accessible from the following website http://frisk.cin.ufpe.br.

2 Background and Example

We focused on posts related to server-side web development, which is a prevalent topic on Stack Overflow [10]. This section provides background on Docker, web development, and shows an example on how Docker can be used to facilitate the reproduction of Stack Overflow posts.

Docker. A Docker *image* is a set of stacked read-only layers; each layer defining a set of file differences that constitute the virtual operating system. A Docker *container* is a running image with a new writable layer on top of the stacked read-only layers, which are loaded from the corresponding image. This top layer is where changes are made on a running container [11]. Interested readers can find tutorials on Docker elsewhere [8], [12].

Web development. Web applications are organized into two parts. The server-side part is mainly responsible for processing the business logic, running cpu-intensive computations, and storing large amounts of data. It handles client requests and generates corresponding answers, potentially in different formats (e.g., JSON). The client-side part is mainly responsible for the user interface, but it can also perform computations and store small amounts of data.

Example. Let us use the Stack Overflow post number 10191048 [13] as an illustrative example. In this case, a Stack Overflow user reports an issue when she tries to start a Node.js server using the Express web framework with the Socket.io library. *Node.js* is a JavaScript runtime environment supporting code execution outside a browser, e.g., execution of server-side code. *Express* is a Node.js-based framework for writing web and mobile applications whereas *Socket.io* is a JavaScript library that supports the implementation of real-time functionality in web applications.

Figure 1a illustrates the code used to report the issue. This code is written in Express. The effect of calling `app.listen(5000)` is to make the web application listen to HTTP/S requests on a given address and port(s) (i.e., localhost). Unfortunately, running this version of the code makes the Node server to launch an HTTP 404 error when the localhost is accessed through the web browser. The root cause

```
var express = require('express')
, app = express()
, io = require('socket.io').listen(app, { log: true });
app.listen(5000);
```

(a) Contents of original app.js file from Stack Overflow post [13].

```
var express = require('express'),
http = require('http');
var app = express();
var server = http.createServer(app);
var io = require('socket.io').listen(server);
server.listen(8000);
```

(b) Proposed solution to the issue in app.js.

```
FROM node:6.9.5
RUN mkdir /app && cd /app
WORKDIR /app
RUN npm install express --save
RUN npm install socket.io --save
COPY . /app
CMD node app.js
```

(c) Proposed Dockerfile to spawn the web service.

Figure 1: Stack Overflow post 7023052—original file, fixed file, and dockerfile.

of the problem in this case is that the developer improperly started the Socket.io. With this code the Socket.io is listen the port before to create a Express/Node.js server, listens to port 5000. Consequently, the server is unable to respond to HTTP requests, producing the error.

Figure 1b presents the proposed fix to the code. In this case, the change recommended by the Stack Overflow developer is to use a standalone HTTP service, as shown in line `var server = http.createServer(app)`. With this method, it is possible to encapsulate the Express object (`app`) in the server object. After the server was created, the developer can connect the Socket.io to that server by using the method `listen()`.

Docker can be used to reproduce the documented issue and proposed solution above. Figure 1c shows a Dockerfile to spawn a web service for the example above. This script loads, with the `FROM` command, the Node 6.9.5 runtime on an Ubuntu image. The script uses the `RUN` command to create the directory `/app` and to change the current directory to it. Then, the command `WORKDIR` sets that directory as the workspace directory on the container. After creating the workspace, the script installs the necessary dependencies using the `npm` package manager. Then, to transfer the code—only `app.js`, in this case—from the host machine to the guest (container), the script uses the command `COPY`. Finally, the command `CMD` specifies the command to be executed when the container is spawned.

Let us start the server, now that we have a dockerfile. Considering our example, the command `docker build -t app $adir` looks for a dockerfile in directory `$adir` on the host machine and builds a corresponding image that will be referred by the name `tse-docker`. Running the command `docker run -p8081:8080 app` creates and runs a container for that image and maps incoming traffic to port `8081` on the host machine to port `8080` on the guest machine, i.e., the container running our server. It is worth noting that developers using FRISK don't need to memorize those command (Section 6.1).

3 Frameworks, Dataset, and Research Questions

This section describes the server-side web frameworks we focused on, the dataset of Stack Overflow posts we used, and the research questions we posed in this study.

3.1 Frameworks

We used GitHub Showcases to identify frameworks for analysis. Showcases is a GitHub service that groups projects by topics of general public interest, providing usage statistics for them. The web framework showcase [14] lists the most popular server-side web frameworks hosted on GitHub according to their number of stars and forks. Note that this list is restricted to projects hosted on GitHub.

Table 1 shows the frameworks grouped by the target programming language. Rows are sorted by the language, number of stars, and number of forks, in this order. We restricted our analysis to a relatively small number of frameworks as the analysis of posts requires human effort. We selected five frameworks that have over 20K stars and over 5K forks. We additionally included `Meteor` as it has the highest number of stars amongst all frameworks. Table 1 shows our selection of six frameworks in gray color. Informally, we found that the selection was consistent with our expectation of popularity of frameworks.

3.2 Questions

To identify questions, we used Data Explorer [15], a service provided by Stack Exchange [16], a network of Q&A forums. The query we used is publicly available [17]. We considered the following selection criteria. (i) We only selected questions tagged with the name of the framework and with the name of the programming language we provided. (ii) We only selected questions not marked as closed. For example, a question can be closed (by the community or the Stack Overflow staff) because it appears to be a duplicate. (iii) We only selected questions that the owner of the question selected a preferred answer.

As the analysis of questions requires human cognizance, we determined a limit of a hundred questions per framework. We prioritized questions in reverse order of their *scores* and extracted the first hundred entries. A similar procedure was adopted in other Stack Overflow mining studies [18]–[22]. The score of a question is given by the difference between the up and down votes associated with the answers to that question. After inspecting the result sets obtained with this method, we realized that some questions, albeit tagged with framework labels, described issues unrelated to the framework itself but related to the programming language it is based on. Considering Rails, for instance, nearly 20% of the questions returned in the original result set was related to Ruby (the language) as opposed to Rails (the framework). To address this issue and complete a set with a hundred questions, we removed those questions and fetched the next questions in the result set.

Table 1: Stats extracted from the GitHub web frameworks showcase [14]. Highlighted rows indicate selected frameworks.

Language	Framework	Stars	Forks	Webpage
Crystal	Kemal	1,273	77	kemalcr.com
C#	Asp.Net Boilerplate	2,138	1,162	aspnetboilerplate.com
	Nancy	4,777	1,185	nancyfx.org
Go	Revel	7,732	1,081	revel.github.io
Java	Ninja	1,575	460	ninjaframework.org
	Spring	11,635	9,155	spring.io
JavaScript	Derby	4,178	240	derbyjs.com
	Express	29,136	5,335	expressjs.com
	Jhipster	5,749	1,291	jhipster.github.io
	Mean	9,714	2,912	mean.io
	Meteor	36,619	4,612	meteor.com
	Nodal	3,940	213	nodaljs.com
	Sails	16,189	1,657	sailsjs.com
Perl	Catalyst	239	96	catalystframework.org
	Mojolicious	1,778	424	mojolicious.org
PHP	CakePHP	6,866	3,108	cakephp.org
	Laravel	28,436	9,392	laravel.com
	Symfony	13,538	5,255	symfony.com
Python	Django	22,822	9,224	djangoproject.com
	Flask	24,291	7,745	flask.pocoo.org
	Frappe´	500	364	frappe.io
	Web2py	1,280	655	web2py.com
Ruby	Hanami	3,487	349	hanamirb.org
	Padrino	2,952	471	padrinorb.com
	Pakyow	722	59	pakyow.org
	Rails	33,910	13,793	rubyonrails.org
	Sinatra	8,553	1,599	sinatrarb.com
Scala	Play	8,754	3,035	playframework.com

3.3 Characterization of Questions

This section describes the dimensions we used to characterize the selected questions: 1) kinds (i.e., what's their purpose), 2) popularity scores (i.e., how popular they were), and 3) prevalence (i.e., how often they appeared in posts).

3.3.1 Kinds

We manually classified our set of 600 posts following two steps. In the first step, two software developers analyzed two disjoint sets of 20 Stack Overflow posts, with the goal of better defining and discussing the codes (i.e., the kinds of posts). After this step, we identified two categories: General and Configuration. The category *general* includes questions related to the presentation of the data or a clarification question about a particular framework feature. The category *configuration* includes questions related to the installation and configuration of the framework. For example, questions about framework misconfigurations (e.g., insufficient privileges to access files and directories). In the second step, each researcher analyzed the rest of the 600 posts independently, followed again by discussion. This step only finished after reaching consensus on the categorization of each post.

After analyzing the questions, developers identified a clear pattern to classify each question. For example, the general questions we analyzed typically follow the pattern "how to implement X in framework Y?". Considering configuration questions, most of the questions (40.15%) follow the pattern "how to fix this issue in framework Y?".

Table 2 shows example questions for each of those categories. For example, the Stack Overflow question 86653 asks how to format a *json* object in Rails using the function `pretty_generate()` from module `json`. As another example, question 17006309 shows how to sort multiple columns in a dataset using the Laravel function `orderBy`. Considering configuration posts, the question 19962736 reports a case

Table 2: Characterization of question kinds.

	Question Id	Question	Answer
general	86653	How can I "pretty" format my JSON output in Ruby on Rails?	Use the pretty_generate() function, built into later versions of JSON.
	17006309	How to use "order by" for multiple columns in Laravel 4?	Simply invoke orderBy() as many times as you need it.
	2260727	How to access the local Django webserver from outside world?	You have to run the dev. server such that it listens on the interface to your network E.g. python manage.py runserver 0.0.0.0:8000
	20036520	What is the purpose of Flask's context stacks?	... This is very handy to implement things like internal redirects.
configuration	19962736	I am trying to run statsd/graphite which uses django 1.6, I get Django import error - no module named django.conf.urls.defaults	Type from django.conf.urls import patterns, url, include.
	11783875	When I run my main Python file on my computer, it works,when I activate venv and run the Flask Python, it says "No Module Named bs4."	Activate the virtualenv and then install BeautifulSoup4.
	19189813	Flask is initialising twice when in Debug mode.	You have to disable the "use_reloader" flag.
	30819934	When I try to execute migrations with "php artisan migrate" I get a "Class not found" error.	You need to have your migrations folder inside the project classmap, or redefine the classmap in your composer.json.
	18371318	I'm trying to install Bootstrap 3.0 on my Rails app. What is the best gem to use? I have found a few of them.	Actually you don't need gem for this, install Bootstrap 3 in RoR: download bootstrap from getbootstrap.com.

where the owner of the question found an error when trying to import module `django.conf.urls.defaults`. In this case, the issue was that the user was using Django version 1.6 which no longer uses that name for the module. The new module name is `django.conf.urls`.

3.3.2 Popularity

We used existing metrics to compare the popularity of the two kinds of Q&A posts we found [23]–[28]. The metrics were the *score* of the question—a number that is adjusted by the crowd according to their appreciation to the question, the number of *views*—a number that increases every time a user visits the question (whether (s)he likes or not), and the number of *favorites*—a number that is adjusted every time a user bookmarks the corresponding question.

We ran tests of hypothesis to evaluate if there was a difference between general and configuration posts w.r.t. these metrics considering our final dataset comprised with 100 Stack Overflow posts of each of six web framework selected in previous step. For a given metric, we propose the null hypothesis that the distributions associated with general and configuration questions have the same median values. The alternative hypothesis being that the corresponding medians differ. As usual, we first used a normality test to check adherence of the data to a normal distribution [29]. According to the Kolmogorov-Smirnov (K-S) normality test, we observed that the data did *not* follow normal distributions. For that reason, we used non-parametric tests which do not assume those distributions. We used two tests previously applied in similar contexts: Wilcoxon-Matt-Whitney and Kruskal-Wallis [29]. The use of an additional test enables one to cross-check results given the inherent noise associated with non-parametric tests. The null hypotheses was not rejected in any test we ran, i.e., the p-values were much higher than 0.05. Considering the metrics we analyzed, there is no statistically significant difference in popularity between general and configuration posts.

3.3.3 Prevalence

Figure 2 shows the distribution of general and configuration questions for each framework. Considering the six frameworks we analyzed and our 600 posts previously categorized,

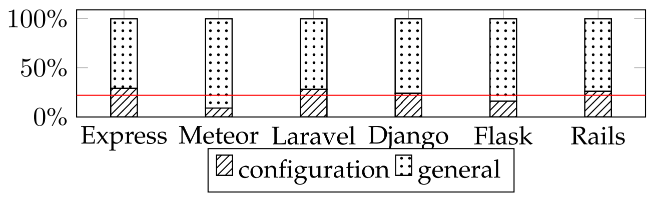


Figure 2: Distribution of general and configuration questions. Horizontal line indicates average value (22%) of configuration questions across frameworks.

it is noticeable that general questions are considerably more prevalent compared to configuration questions. It is also noticeable that Meteor manifests the lowest proportion of configuration questions to general questions. That happens because Meteor, in contrast to alternative frameworks, provides preconfigured options and a rich set of libraries built-in.

3.4 Research Questions

We pose the following questions:

- Survey
  - **RQ1.** What are the perceptions of Stack Overflow users towards the use of Docker to reproduce posts?
- Exploratory Study
  - **RQ2.** How difficult is it for developers with elementary training in Docker to dockerize Stack Overflow posts?
  - **RQ3.** How big and similar are dockerfiles?
  - **RQ4.** How often can developers dockerize posts?
- User Studies
  - **RQ5.** Do Stack Overflow users access and reproduce posts using Frisk?

We ran different experiments to assess the feasibility of using container technology (e.g., Docker) to assist Q&A forum users. The Survey captures the perceptions of developers about the usage of container technology. The exploratory study evaluates the main concerns raised by developers during the survey. Finally, the user study evaluates the proposed idea with real developers using FRISK, a prototype system we developed to support the integration of Docker in Stack Overflow.

4 Survey

This section describes a survey we conducted with the goal of evaluating the perceptions of Stack Overflow users towards the use of Docker for reproducing posts.

4.1 RQ1: What are the perceptions of Stack Overflow users towards the use of Docker to reproduce posts?

To participate on this survey, we selected active users of five of the six frameworks from Figure 2. We discarded Meteor as there is no Docker support for it as of the day of the writing. Our method to select participants was as follows. For each framework, we selected 1K users with the best reviewing scores to posts. We could not send invitations to all 5K users (1K users * 5 frameworks) as Stack Overflow does not allow users to publish e-mails on their pages. As such, we attempted to establish links between Stack Overflow and GitHub accounts to find their emails. Using this approach, we identified a total of 1,548 potential participants from a total of 5K users. Finally, we sent out invitations for a survey. Participants were encouraged to complement their answers to questions with text describing their impressions on using Docker to reproduce Stack Overflow posts. This part was not mandatory and few participants provided additional information. The survey questions are as follows.

1) Are you familiar with Docker? (a) Never heard of it; (b) Have interacted with it a bit; (c) Use it frequently.
2) Do you think executable Dockerfiles could help developers understanding Q&As from Stack Overflow? (a) Yes; (b) No; (c) I don't know.
3) What do you think are the main challenges in using Dockerfiles at Stack Overflow? (a) Security concerns; (b) It is time consuming to read and write dockerfiles; (c) Lack of sysadmin skills; (d) Most Q&As are pretty straight-forward; (e) I don't know.

For the first question, the intuition is that it would be challenging to incentivize adoption if familiarity with the technology was very low. The second question assesses the perceived utility of our proposal. Finally, the third question evaluates technical concerns of users about dockerization at Stack Overflow. A total of 106 users answered the survey. Of which, we discarded 13 invalid answers (e.g., auto-reply answers). It is worth noting that not every participant answered all questions. For example, someone that answered "a" to the first question would not answer the remaining questions. However, most participants answered most questions. Table 3 shows the distributions of the answers to the three questions. The cells in gray color highlight the choices with the highest amount of answers for a given question.

Considering question one, we found, with some surprise, that ∼90% of participants who answered the survey were familiarized with Docker and a large proportion of them (35.5%) use Docker frequently.

Considering question two, 39.2% of the participants were optimistic about using Docker to reproduce Q&A posts. Participants in this group mentioned that Docker would help to reproduce complex environments and version-pinned questions. It is worth mentioning that most of those participants (95% of them) were familiar with Docker (i.e., answered "b" or "c" to question one). However, the chart also shows that 21.6% of the participants do not think that Docker would help. For example, some developers of the Express framework commented that, when the post did not depend on server-side features, Docker would not be necessary.

Table 3: Distribution of answers to survey questions.

	Answers				
	a	b	c	d	e
Question 1	9.7%	54.8%	35.5%	-	-
Question 2	39.2%	21.6%	39.2%	-	-
Question 3	12.6%	32.3%	15.0%	33.1%	7.1%

Considering question three, developers pointed to effort (option"b") and need (option "d") in 32.3% and 33.1% of the answers, respectively. Despite the optimism signaled by developers, a large proportion of them answered that reading and writing dockerfiles could be time-consuming and posts could be either straight-forward or not require fully-functioning code for understanding. Participants that selected option "c" commented that creating dockerfiles could be challenging to new developers. A relatively smaller proportion, but still significant, of the participants (12.6%) mentioned concerns about security (option "a"). However, none of them specified the reason why.

Answering RQ1: A total of 39.2% of the participants in the survey believe that Docker could help Stack Overflow users address reproducibility issues. The biggest concern among the 21.6% of the participants who were skeptical are cost and need.

5 Exploratory Study

To better understand the actual issues in creating containers for Stack Overflow posts, we designed an exploratory study where the first author of this paper and another developer created containers for a selection of posts.

Originally, the concerns we considered in the study were *need*, *cost*, and *realizability*. However, prior work already reported findings related to need so we focused on the later two concerns. Treude et al. [30] and also Beyer and Pinzger [31] found that developers often find solutions to answers to general posts by looking at documentation and tutorials. Informally, we confirmed the same findings during the experiment we conducted to answer RQ4. However, we found that, despite demanding less changes to code artifacts (see Section 5.2), configuration posts are more involved than general posts and would benefit of reproduction scripts. Therefore, considering the concern need, our conclusion is that containers are helpful to reproduce configuration posts.

The exploratory study described in this section focused on *cost*, the top concern raised by the survey's participants, and *realizability*, i.e., the ability to translate natural language posts into containers. Research questions RQ2 and RQ3 relate to the concern cost whereas question RQ4 relates to realizability.

5.1 RQ2: How difficult is it for developers with elementary training in Docker to dockerize Stack Overflow posts?

The goal of this research question was to evaluate the ability of developers to create containers from Q&A posts in a pessimistic scenario. This experiment involved students from a grad-level Software Testing course at the authors' institution. No student in the class had previous experience with Docker but most of them have heard recently about it. We dedicated a 2h in-lab class to train students—1h for Docker and 1h for the basics on server-side web development. Given the limited time, we restricted the training to Flask for its simplicity and familiarity of many students with Python. All students had access to a similar desktop computer. After the training session, students met twice to run the actual experiment. The activity was realized in class with the supervision of the first and third author of this paper.

We assigned each student the task of reproducing five Stack Overflow posts. We used the first 30 minutes of the class to instruction. After that, we asked students to prepare the scripts and a short critique of the approach by e-mail writing pros and cons. They had 90 minutes maximum for that. To guarantee the correctness of each script reproduced, we used the answer selected by the original poster of the question as ground truth.

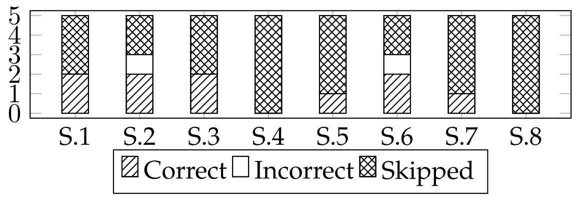


Figure 3: Students' performance in preparing dockerfiles.

Figure 3 shows a bar plot indicating the performance of the students enrolled in the class. Two of the eight participants did not submit an answer (S.4 and S.8). Four participants submitted two correct answers, and two submitted one correct answer. The main reasons reported by students for not being able to reproduce an issue were 1) lack of knowledge in the language or the framework and 2) incomplete excerpts of code in Q&A posts. Students firmly indicated in their reports that the training session on Docker was enough for the assignment, but they felt inexperience in the target programming language and framework.

Answering RQ2: External factors (e.g., language experience, incomplete excerpts, and interest of students), which are hard to control, clearly affected results. However, still, we believe that interested students performed relatively well in the assignment.

5.2 RQ3: How big and similar are dockerfiles?

We used size and similarity of dockerfiles as proxies of complexity with the goal of better estimating human cost. This section elaborates on our findings. Figure 4 shows tables detailing statistics on these metrics.

Table 4: Size and similarity of dockerfiles.

		Same	Size (LOC)	Sim.
General	Express	48.8%	6.6	90.95%
	Laravel	100%	12.0	100.00%
	Django	41.1%	11.9	93.63%
	Flask	47.5%	11.4	96.38%
	Rails	55.0%	15.4	92.44%
Configuration	Express	42.9%	6.4	92.39%
	Laravel	84.2%	11.7	95.50%
	Django	57.1%	11.1	92.39%
	Flask	84.0%	13.2	96.78%
	Rails	75.0%	15.3	95.07%

(a) Number of cases dockerfiles are identical (Same), Average size of dockerfiles (Size), and average similarity of dockerfiles (Sim.). Table 5 shows the absolute numbers of questions for each pair of framework and category.

		# Files	Churn	#Ins.	#Mod.	#Del.
General	Express	1.5	9.4	3.8	5.5	0.1
	Laravel	3.7	25.4	18.6	4.7	2.1
	Django	3.9	20.1	18.3	1.8	0.0
	Flask	1.6	8.7	5.7	2.9	0.1
	Rails	8.0	22.1	21.8	0.2	0.1
Configuration	Express	1.2	9.9	4.0	4.9	1.0
	Laravel	1.8	6.8	5.3	1.3	0.2
	Django	2.4	3.5	2.0	1.5	0.0
	Flask	1.6	4.7	2.5	1.8	0.4
	Rails	1.0	3.2	3.0	0.2	0.0

(b) Application artifacts (e.g., source and config. files) modified in boilerplate code when preparing containers.

Table 4a shows results grouped by frameworks. Column "Same" shows the percentage of cases where the dockerfile was identical to the reference file (see Section 5.3). In those cases, the developer only changed application files (e.g., source and configuration files) to run the container. Columns "Size" and "Sim." show, respectively, size and similarity of dockerfiles of a given framework. Size refers to the average size across all dockerfiles whereas similarity refers to the average across all pairs of dockerfiles. We used the Jaccard coefficient [32] to measure string similarity. We did *not* embed application code within dockerfiles.

Note that in many cases it was unnecessary to modify the reference dockerfile to reproduce the post. Laravel was an extreme case: all 40 scripts from the general category for this framework were identical to the reference dockerfile; changes were made only in application files. This peculiar case happens because, for some frameworks, including Laravel, the corresponding boilerplate project comes with a built-in package manager [33] that resolves dependencies on-the-fly. For frameworks other than Laravel and Express, the number of identical dockerfiles was smaller for general posts than it was for configuration posts. The typical reason for these cases is that the dockerfile includes instructions to create a database that is necessary to reproduce the post.

Considering size, results shows that dockerfiles are typically very short, ranging from a minimum of 6.6LOC in Express to a maximum of 15.4LOC in Rails. Besides, the size of dockerfiles for Express is significantly smaller compared to other frameworks. That happens because the Docker official image of Node.js [34], which Express builds on, comes with a fairly complete set of packages that an application needs to run. Finally, results show that dockerfiles are very similar to each other with an average similarity score above 94%.

Table 4b reports the number of changes made in application files relative to the boilerplate code we used as a reference to create new containers. These files do *not* include the dockerfile. Column "# Files" shows the average number of files modified or created relative to the reference code. Column"Churn" shows code churn as the amount of lines added, changed, or deleted while reproducing the post. Columns "#Ins.", "#Mod." and "#Del." specify the kind of change. All reproduced posts modified at least one application file and the changes were higher in general posts compared to configuration posts. In configuration posts the change was typicall a minor tweak to fix the problem whereas general post required more complete descriptions.

Answering RQ3: Results indicate that dockerfiles to reproduce posts are typically very small and similar to each other. Considering application files, general posts change more files and make more changes on each file compared to configuration posts.

5.3 RQ4: How often can developers dockerize posts?

The goals of this research question are i) to estimate the number of posts that can be translated into executable scripts and ii) to understand the reasons that prevent the creation of scripts.

As Mondal et al. [4], we involved two software developers to carry out the task of writing dockerfiles to the 600 posts from our dataset. Both developers had three years of professional experience in web development. One developer had working experience with JavaScript and another developer, the first author of this paper, had working experience with Laravel (PHP) and Django (Python). To create containers, we used a Debian 8.6 Jessie machine [35] with `docker` and `docker-compose` [8] installed.

The task of writing a dockerfile for a given post consists of the following steps: (1) understand the post, (2) reproduce the post on the developer's host machine, (3) create the dockerfile, and (4) spawn the container and check correctness according to the instructions in the post. For general post (see Section 3.3), developers were asked to produce one dockerfile with the solution to the question. For configuration posts, developers were asked to produce two dockerfiles: one to reproduce the issue and another to illustrate the fix. Developers used stack traces, when available in the posts, to validate the correctness of their scripts. For example, if the post reports an issue, the developer used the trace to validate both the "issue" script and the corresponding "repair" script for the presence (respectively, absence) of the manifestation in the trace. Developers also validated each other's containers for mistakes. While preparing those reproduction scripts, the two developers noticed that the files they produced were very similar. For that reason, they prepared per-framework template files to facilitate the remaining work. For dockerfiles, this task was manual. For application code, three of the frameworks used—Django, Laravel, and Rails—already provide tools to generate boilerplate code.

Table 5: Breakdown of problems found while generating dockerfiles. Column "Σ-P*" indicates the total number of posts reproduced per framework.

			Unreproducible				Costly		
		Σ	P1	P2	P3	P4	P5	P6	Σ-P*
General	Express	71	-	1	26	1	-	-	43
	Meteor	91	91	-	-	-	-	-	0
	Laravel	72	-	17	13	2	-	-	40
	Django	76	-	5	12	8	-	-	51
	Flask	84	-	2	19	5	-	-	58
	Rails	74	-	-	32	-	2	-	40
	Total	468							232
Configuration	Express	29	-	12	-	-	1	-	16
	Meteor	9	9	-	-	-	-	-	0
	Laravel	28	-	9	-	-	-	6	13
	Django	24	-	8	-	-	7	3	6
	Flask	16	-	4	-	-	-	-	12
	Rails	26	-	11	-	-	1	5	9
	Total	132							56

As expected, some posts were *not* reproduced because they were unreproducible or because they were too expensive to reproduce. Table 5 shows the breakdown of those problems per framework and category. Column "Σ" shows the total number of posts associated with a given framework. Columns "P1-P6" show the number of posts that were *not* reproduced due to a given problem. Column "Σ-P*" shows the total number of posts developers reproduced with Docker using the setup we described. A dash indicates that no problem of a certain kind was found for a given framework.

The problems developers found are as follows: *P1 (Unsupported):* A feature necessary to dockerize the post is still unsupported. For example, as of this date, Docker does not support a particular feature from `tar` necessary to run Meteor [36], [37]. *P2 (Lack of details):* The question lacks important details to reproduce the problem (e.g., post 26270042). *P3 (Conceptual):* The question is a conceptual question about the framework usage (e.g., post 20036520). *P4 (Clarification):* The question is a clarification question about the framework (e.g., post 14105452). *P5 (User interaction):* Console interaction is necessary to create a container (e.g., post 4316940). *P6 (OS-specific):* The post is specific to a non-Linux OS (e.g., post 10557507).

It is important to note that, given our limited resources, we decided to restrict our study to posts that could be reproduced without console interaction and to posts that are specific to Unix-based distributions. Consequently, problems P5 and P6 can be addressed. Only a small fraction of posts (4.1%) did not satisfy these two constraints.

We also note that a good number of posts (69) were not reproduced because the description was unclear (P2). We did expect that textual descriptions could lead to this problem, but still we were surprised by the considerable number of cases, 11.5% of the total. Overall, developers translated 49.6% of the general posts and 43.2% of the configuration posts. If we remove from these counts posts that are, in principle, reproducible (P5 and P6), we increase those numbers to 49.8% and 52.7%, respectively. If we discard conceptual posts (P3), the numbers of general posts reproduced becomes 63.4%. If we discard unclear posts (P2), the numbers of configuration posts reproduced becomes 63.6%.

Answering RQ4: Without soliciting additional information to posters, a total of 48% of the posts could be dockerized. Problem P2 (lack of details) affected configuration posts relatively more compared to general posts. This is evidence of the importance of reproducibility in this context.

6 User Study

This section presents the user study we conducted on Frisk, a tool we developed to enable rapid creation, modification, and sharing of Docker containers. The goal of Frisk is to facilitate the developer task of solving Stack Overflow issues to server-side development problems.

Section 6.1 describes the basic functionalities and organization of Frisk. Section 6.2 describes the design of the user study, e.g., it describes how we contacted Stack Overflow users to validate Frisk's usability. Finally, Section 6.3 reports results showing how developers interacted with Frisk.

6.1 Frisk

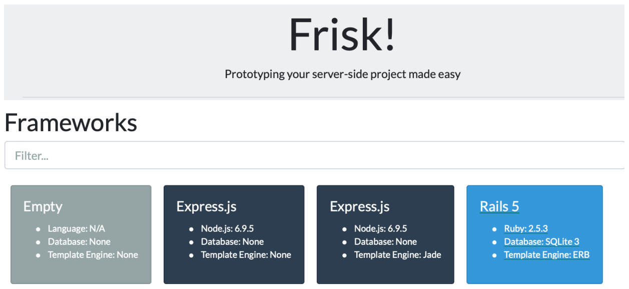
Frisk is a prototype tool designed to help developers with minimal background on Docker to create, maintain, and share containers. Our conjecture is that a tool like Frisk could be useful to Stack Overflow by linking Frisk's sessions to posts. The tool is available online[1] and does not require user authentication as to encourage developer's adoption. Similar rationale is used in JSFiddle [38], a system to facilitate front-end development. A tutorial of Frisk is available online [39].

6.1.1 *Main functionalities*

Figure 4a shows the home page of Frisk. A user should notice two features: 1) there are no login credentials to access the system and 2) the access to Frisk is made upon the selection of a card that defines what needs to be included in the container (e.g., library or runtime). If the desired option is not available, the user can select the empty card and configure the dockerfile himself.

Figure 4b shows the UI of the main page of Frisk. For space, we did not show the left pane of the screen showing virtual machines associated with the session. This session only contains one virtual machine. This screen is produced when the user selects the third template card (Express.js) on Figure 4a. By selecting that template, Frisk creates a dockerfile declaring all dependencies declared in the card. For example, that card includes a dependency for the Jade template engine [40] that translates page descriptions into HTML. Frisk enables modifications in the dockerfile and the boilerplate code associated with the template. Observe the treeview on the top right corner of the image.

The screen is divided into three panes. The left pane—omitted for space—shows the created virtual machines for the current session and a button to create new ones. (Additional virtual machines can be be useful to prototype microservices.) By default, one virtual machine is created on session entry. The other panes focus on one selected machine, which is highlighted on the left pane. The central



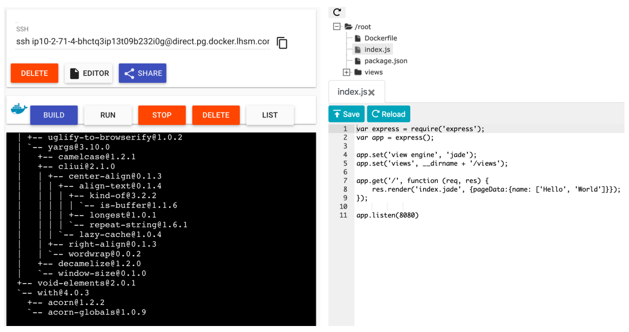
Empty
- Language: N/A
- Database: None
- Template Engine: None

Express.js
- Node.js: 6.9.5
- Database: None
- Template Engine: None

Express.js
- Node.js: 6.9.5
- Database: None
- Template Engine: Jade

Rails 5
- Ruby: 2.5.3
- Database: SQLite 3
- Template Engine: ERB

(a) Home page.



```
ssh ip10-2-71-4-bhctq3ip13t09b232i0g@direct.pg.docker.lhsm.cor
```

```
│ +-- uglify-to-browserify@1.0.2
│ `-- yargs@3.10.0
│   +-- camelcase@1.2.1
│   +-- cliui@2.1.0
│   │ +-- center-align@0.1.3
│   │ │ +-- align-text@0.1.4
│   │ │ │ +-- kind-of@3.2.2
│   │ │ │ │ `-- is-buffer@1.1.6
│   │ │ │ +-- longest@1.0.1
│   │ │ │ `-- repeat-string@1.6.1
│   │ │ `-- lazy-cache@1.0.4
│   │ +-- right-align@0.1.3
│   │ `-- wordwrap@0.0.2
│   +-- decamelize@1.2.0
│   `-- window-size@0.1.0
+-- void-elements@2.0.1
`-- with@4.0.3
  +-- acorn@1.2.2
  `-- acorn-globals@1.0.9
```

```
var express = require('express');
var app = express();

app.set('view engine', 'jade');
app.set('views', __dirname + '/views');

app.get('/', function (req, res) {
    res.render('index.jade', {pageData:{name: ['Hello', 'World']}});
});

app.listen(8080)
```

(b) Main page.

Figure 4: Frisk's UI.

pane provides a command shell and buttons to interact with the selected machine whereas the right pane shows a simple editor for the files created by the template. In contrast to changes in dockerfiles, updates made directly on the shell from the command line (e.g., using `apt-get install`) are *not* persistent. Without these templates, users would have to prepare dockerfiles and source files from scratch.

A typical Frisk scenario of use consists of selecting a template, modifying necessary files on the editor panel (at the right of the screen), clicking the *build* button to create a Docker image, clicking the *run* button to spawn the corresponding Docker container (referring to the image created last in the session), and, finally, clicking the *share* button to generate a URL for the session. The original fork of "Play With Docker" does not implement these buttons. We decided to create them to improve usability.

When a user accesses the URL created with the share button, Frisk creates a copy of the corresponding files and creates a container to isolate that session from other users, who could modify the corresponding containers however they want in their own sessions. Using these URLs, Stack Overflow users can recover Frisk sessions and visualize solutions to posted issues.

6.1.2 *Implementation details*

Frisk is implemented as a fork of "Play With Docker" [41], [42], a system recently sponsored by Docker Inc. to train people on Docker. Compared with "Play With Docker", the main differences of Frisk are 1) the ability to share sessions, 2) the ability to bootstrap sessions from templates, and 3) the availability of a toolbar including buttons to run Docker commands with default parameters. We noticed while running this user study that changing those parameters was rarely necessary. Consequently, users can use the tool

1. http://frisk.cin.ufpe.br

Table 6: Data obtained from FRISK analytics.

Framework	Duration	#Sessions	Builds	Runs	Accesses
Django	13m41s	90	62.22%	51.11%	17.78%
Express	9m49s	90	68.89%	58.89%	55.56%
Flask	9m59s	175	86.86%	74.86%	49.14%
Laravel	11m26s	105	87.62%	74.29%	48.57%
Rails	11m38s	103	86.41%	54.37%	50.49%
total.	56m33s	563	-	-	45.29%
avg.	11m19s	112	78.40%	62.71%	44.30%

without much knowledge on Docker commands. It is also worth noting that FRISK runs as a Docker container and the containers created with it run inside that container. Consequently, the concern that 12.6% of participant raised on question 3(a) of the survey has no basis as our tool runs on an isolated environment on our cloud servers.

6.2 Study Design

This section elaborates on the design of a user study to assess the willingness of Stack Overflow users in adopting FRISK, which could, conceptually, be used to link Docker containers to Stack Overflow.

We initially considered the idea of asking developers to prepare FRISK sessions, but, we realized people would likely be discouraged to participate. Although we believe the effort to prepare sessions would not be high, we thought people would have no clear incentives for doing that work on a system they did not know. Instead, we planned to ask people to evaluate FRISK sessions that we created for the Stack Overflow posts they created. The rationale is that developers would relate to their work, and they could play from a real example that they could modify. Thus, we created FRISK sessions for previously-created posts, sent e-mails to developers and added comments to posts as to advertise the FRISK solution, and then monitored user activity.

We prepared FRISK sessions for a selection of configuration-related posts. Each session reproduces the preferred answer to a given Stack Overflow question. In total, we prepared 100 FRISK sessions, 20 for each framework. The rationale was to identify experienced developers interested in the high-quality posts they were involved. For these posts, we contacted the question makers and respondents.

Our initial attempt to advertise the FRISK session was to edit the preferred answer on Stack Overflow adding a link to the FRISK session. Unfortunately, we realized after-the-fact that the Stack Overflow policy rejects posts that may look like a tool advertisement. As a consequence, the updates we created were dismissed by the Stack Overflow community. To address that, we contacted developers through e-mails and comments in Stack Overflow. In both cases we provided a link to the FRISK container, explained what it offered, and asked people to try the tool. For the comments, we did not name the tool as to prevent rejection of the post.

6.3 Do Stack Overflow users access and reproduce posts using Frisk?

This research question evaluates how Stack Overflow users interact with FRISK. We used proxy metrics to measure interest of users. Table 6 summarizes results, broken down by framework. Column "Duration" shows the average time users spent interacting with FRISK. The period of interaction begins from the point the user accesses the URL–created to share the session–and stops at the moment of the last interaction–we looked for inactivity in the logs. Column "#Sessions" shows the number of sessions accessed for a particular framework. Columns "Builds", "Runs", and "Accesses" show, respectively, the percentage of cases (i.e., fraction of number from column "#Sessions") where users clicked the *build* button, the *run* button, and the link generated to access the running service on the browser. Note that the percentages must not increase as one can only run a container if she has built the image and one can only access the service if she has ran the container.

We found surprising the interest of developers in Flask, given that this is the least popular framework among the five we selected [43]. Looking at column "Accesses", one can observe that a total of 255 accesses were made. We were also surprised that Django, which is another Python very popular framework in this group, was the case with the lowest rate of accesses by developers. We conjecture that the amount of training in a given framework influenced the number of successful accesses, which is our proxy for interest in FRISK. Recall that the participants are trained developers who prepared answers to posts we dockerized.

Finally, we noticed a relatively high gap between columns "Runs" and "Accesses", provided that to access the service–and count one access–FRISK users only needed to click on a link after spawning a container. Observe the values of these columns on the row for Laravel. One possible reason for that is that users are missing the URL link to make an HTTP request to the running service. This link is dynamically created after the container starts to run.

We observed from these results that developers interacted with the system for a reasonable amount of time ($\sim$10m). FRISK received a substantial number of user accesses over the period of a month and, considering the number of posts we advertised (563), many of these accesses resulted in an access of the corresponding web service ($\sim$45%).

Of the 563 FRISK sessions we created for Stack Overflow posts, $\sim$45% resulted in an access to the corresponding web service that required the user going through a sequence of steps in the tool.

Overall, we believe that the observations made during this experiment provides early evidence on the interest of the community in FRISK as a tool that could be used to link Docker and Q&A forums, such as Stack Overflow.

7 THREATS TO VALIDITY

The main threats to validity of this study are the following.

External Validity. The extent results can be generalized is limited by our dataset, which includes Q&A posts from a selection of web frameworks. In principle, there could be frameworks and posts with different characteristics that could lead to different findings. To mitigate those issues, we selected the six most popular frameworks, according to a recent showcase from GitHub and questions described in

Section 3.2. It remains to evaluate the extent to which our observations would change when using different frameworks (e.g., frameworks not in the listing from Figure 1) and different criteria for selecting questions for each framework.

Another threat is related to the generalization of the templates prepared in this study. In principle, there could be scripts unfit to those templates, i.e., scripts that would require significant changes. Another threat is related to the number of cases we used to build the template. Developers considered a relatively small number of circumstances for preparing those scripts and validated them against a large number of scripts.

Internal Validity. Our results could be influenced by unintentional mistakes made by humans involved in this study. For example, students were involved in a user study, whereas developers manually categorized questions in difficulty levels and elaborated dockerfiles. All those tasks could introduce bias. We used Card Sorting [44] to mitigate the problem of incorrectly categorizing questions. To make sure that the scripts were correct, developers were instructed to strictly follow the preferred Q&A post answers to reproduce similar problems.

We also encouraged developers to do their best to reproduce as many questions as possible, noting that one of the authors was one of the developers. As for the answer of students in the user study, we analyzed their responses carefully, comparing them with the solution prepared by the instructors. It is important to note that all artifacts produced during this study are publicly available for scrutiny [45]. Finally, the monitoring infrastructure that we used for tracking FRISK usages did not take into account the possibility of a user accessing the same session multiple times. However, we manually analyzed the logs and did not notice a high number of accesses for individual FRISK containers, suggesting that that was not an issue.

Construct Validity. We considered several metrics in this study that could influence some of our interpretations. For example, we used metrics of document similarity to assess how (dis)similar the dockerfiles produced by developers are. To mitigate the bias associated with the metric selection, we used multiple metrics and confirmed that the similarity was very high as not to compromise corresponding conclusions.

8 DISCUSSION

Our results suggest that the fears developers manifested during our survey (Section 4) were not all justified. Developers mentioned concerns of cost in writing dockerfiles, but that task has shown to be short. The artifacts involved in a post are similar to each other (Section 5 RQ3) and that enabled the construction of templates–including reference dockerfiles and boilerplate code–that allows developers to be more productive in this task.

Developers also manifested concerns about need of using Docker in that context. We found that was the case for the posts in the general category. However, there is an important group of post for which solutions are non-trivial, and integrating Docker could be helpful. The study of Horton and Parnin [46] corroborates that. Many code snippets they analyzed from GitHub required non-trivial configuration-related changes to be executed, including missing dependencies, misconfigured files, reliance on a specific operating system, or some other environmental issue. In general, our results were also consistent with the Mondal et al. [4], since we also found a similar number of posts that are reproducible (around 60%). Finally, developers also manifested concerns with security, but FRISK containers run on the cloud so compromising the user space is not possible.

Overall, our conclusion is that the use of Docker should be encouraged for configuration-related posts, which are those that appear to be more technically involved. Note that although FRISK is a working prototype, there are many features that can be added to the tool to make it more useful. For example, we plan to extend FRISK in the near future with additional features such as improved text editor and debugging support. However, we believe that our results provide early evidence of 1) the usefulness of FRISK and 2) the important of linking Docker and Stack Overflow.

9 RELATED WORK

We organized related work in two groups–work related to educational tools and collaborative IDEs and work related to mining repositories.

9.1 Educational tools and Collaborative IDEs

Tools such as Repl.it [47] and JSFiddle [38] provide support to create and share self-contained code examples. These platforms are great for teaching, but they are not well suited for the creation of complex environments, including databases, web servers, etc. The configuration posts that we analyzed in this paper involve at least one or more of these aspects. Collaborative IDEs, such as Cloud9 [48] and CodeAnywhere [49], can, in principle, build more complete local environments but these are private, making sharing more difficult. It is important to note that exploring live collaboration seems an important feature to have in this context that should be explored in FRISK.

9.2 Mining repositories

We elaborate below work that reports on issues in repository data and work that proposes ways to fix those issues.

Recent work studied various aspects of development behavior manifested through Stack Overflow data. For example, Yang et al. [1] criticized Stack Overflow code quality, indicating that code is written mostly for illustrative purposes and "compilability" is not typically considered. Terragni et al. [2] and Balog et al. [3] also found that compilation issues are common. Bajaj et al. [24] analyzed Stack Overflow questions to understand common difficulties and misconceptions among JavaScript developers. They focused on a restricted domain; in their case JavaScript, in our case server-side frameworks. In a different study, Treude et al. [30] found that often answers to questions become a substitute for official documentation.

Considering the general category of questions, the results we found are consistent with theirs. Allamanis and Sutton [50] automatically analyzed arbitrary Stack Overflow questions using standard data mining techniques. In contrast to them, we explored a narrower domain and involved humans in the analysis of questions. Beyer and Pinzger [31] presented an automatic approach to classify documented

Android issues in Stack Overflow using the Apache Lucene search engine [51]. They used manual classification of questions using Card Sorting as we did but for a different reason–to build the ground truth to base the computation of accuracy of automatic classification techniques. The idea is complementary to ours. Searching for good post candidates for creating containers is could help engage developers in using FRISK. Yang et al. [52] automatically analyzed code snippets from Stack Overflow to measure how often these snippets originate from open source projects. They found that in many cases the link could be recovered. One interesting avenue of future work is to slice minimal FRISK containers from those projects.

Recent work proposed solutions to existing problems in Stack Overflow or GitHub. For example, Terragni et al. [2] proposed CSNIPPEX, a technique to automatically transform Stack Overflow code snippets into compilable Java code. Their technique looks for fixes to compilation errors, such as missing import declarations. More recently, Horton and Parnin [46] proposed Gistable, a tool to automatically transform Python code snippets from GitHub into runnable Dockerfiles. As CSNIPPEX, their tool also makes simple transformations, if necessary, to repair the Gist code. Differently from CSNIPPEX, Gistable tries to write Dockerfiles from a given Stack Overflow post, creating a large database of Dockerfiles based on real-world questions. Horton and Parnin [53] improved Gistable proposing DockerizeMe, a technique for inferring the dependencies needed to execute a Python code snippet without import error. In contrast to Gistable, FRISK provides an infrastructure for sharing solutions and focuses on problems (or solutions to those problems) that may require multiple files and services (e.g., database, templates) to demonstrate those problems whereas Gistable focuses on compiling self-contained snippets. Finally, Balog et al. [3] proposed DeepCoder, a technique that uses Deep Learning to synthesize code from Stack Overflow code snippets. In principle, DeepCoder could capitalize on better code snippets to improve code synthesis. These works provide evidence on the importance of writing quality code at Q&A forums. Note, however, that high-quality code alone is insufficient to demonstrate certain kinds of issues. This is noticeable on the configuration questions mentioned in this paper. Executable scripts can help on that.

9.3 Studies about the Docker Ecosystem

Different aspects of the Docker ecosystem such as understanding the robustness of Dockerfiles and its adoption in Software Engineering were recently studied. For example, Cito et al. [6] conducted an exploratory study to characterize issues found in dockerfiles. The study found that most common issues (28.6%) arise from missing version pinning, and 34% of 560 Dockerfiles checked were not built correctly. This study also sheds light on the reproducibility problem we focused on this paper, but they focused directly on dockerfiles as opposed to their use to address server-side questions. Recall that FRISK creates dockerfiles from templates. Consequently, unless the developer needs to customize dependencies, version-pinning issues would not be in effect as the files would be likely consistent.

Hassan et al. [54] proposed RUDSEA, an approach to help developers update container image configuration files more efficiently, while Zhang et al. [7] conducted a large-scale study on Docker Hub/GitHub to comprehend the use of container technology in Continuous Integration workflows. Although the focus of these studies are different from ours, their findings indicate that the use of tools like Docker is important in Software Engineering.

10 CONCLUSIONS

This paper reports on a study to asses the feasibility of using Docker to reproduce Q&A posts. We focused on posts related to server-side development, which is an important application domain [10]. This is a timely and important problem given the constant pressure for increased productivity in this domain [55] and the observation that web developers heavily rely on Q&A [56] forums nowadays.

Our results show that reproduction scripts would help the most to address configuration posts and that the presence of this kind of post is not uncommon. Furthermore, we observed that Stack Overflow users support the idea to use container technology to facilitate the reproduction of a post.

We found that the task of preparing containers is not very costly, especially when there is automated tool support. We developed a tool, named FRISK, to facilitate that step. The tool enables Stack Overflow users to create and share containers. In summary, our results provide early evidence that the integration of reproduction scripts (e.g., Docker scripts) in Q&A forums (e.g., Stack Overflow) should be encouraged in certain cases, such as those involving configuration posts.

As future work, we want to investigate the use of FRISK in other contexts, including training students, training professionals in new technologies, and outsourcing debugging activities. We also want to add new features to the tool as cooperative edition of FRISK sessions, language-specific editors with auto-complete support, and a debugger.

ACKNOWLEDGMENTS

This research was partially funded by INES 2.0, FACEPE grants PRONEX APQ 0388-1.03/14 and APQ-0399-1.03/17, CAPES grant 88887.136410/2017-00, and CNPq grant 465614/2014-0. Luis was supported by the FACEPE/CAPES fellowship number PBPG-1175-1.03/16.

REFERENCES

[1] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: an analysis of stack overflow code snippets," in MSR. ACM, 2016.

[2] V. Terragni, Y. Liu, and S.-C. Cheung, "Csnippex: Automated synthesis of compilable code snippets from q&a sites," in ISSTA, 2016, pp. 118–129.

[3] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs," CoRR, vol. abs/1611.01989, 2016.

[4] S. Mondal, M. M. Rahman, and C. K. Roy, "Can issues reported at stack overflow questions be reproduced?: An exploratory study," in MSR, 2019, pp. 479–489.

[5] Sysdig. 2018 docker usage report. https://sysdig.com/blog/2018-docker-usage-report/.

[6] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An empirical analysis of the docker container ecosystem on github," in MSR, 2017, pp. 323–333.

[7] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: An empirical study of containerized continuous deployment workflows," in ESEC/FSE, 2018, pp. 295–306.

[8] Docker. (2017) Docker website. https://www.docker.com/.

[9] L. Melo and M. d'Amorim. (2019) Paper artifacts. https://docker-so-study.github.io/.
[10] (2019) Stack Overflow Developer Survey Results. https://insights.stackoverflow.com/survey/2019.
[11] (2017) Docker engine documentation. https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/.
[12] Docker cheat sheet. https://goo.gl/Lq93kV.
[13] S. User. (2017) Express - configure node.js, express and socket.io as unique service. https://stackoverflow.com/questions/10191048/.
[14] GitHub. (2017) Web application frameworks server-side showcase. https://github.com/showcases/web-application-frameworks.
[15] S. Exchange. (2017) Stack Exchange Data Explorer website. http://data.stackexchange.com/.
[16] ——. (2017) Stack Exchange website. http://stackexchange.com/.
[17] Anonymous. (2017) Dataexplorer q&a selection query. https://data.stackexchange.com/stackoverflow/query/621859.
[18] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu, "Want a good answer? ask a good question first!" CoRR, vol. abs/1311.6876, 2013. [Online]. Available: http://arxiv.org/abs/1311.6876
[19] Y. Yuan, T. Hanghang, X. Feng, and L. Jian, "Predicting long-term impact of cqa posts: a comprehensive viewpoint," in SIGKDD, 2014.
[20] Z. Yanzhen, Y. Ting, L. Yangyang, M. John, and Z. Lu, "Learning to rank for question-oriented software text retrieval," in ASE, 2015, pp. 1–11.
[21] Y. Yuan, T. Hanghang, X. Tao, A. Leman, X. Feng, and L. Jian, "Joint voting prediction for questions and answers in cqa," in ASONAM, 2014, pp. 340–343.
[22] Y. Ting, X. Bing, Z. Yanzhen, and C. Xiuzhao, "Interrogative-guided re-ranking for question-oriented software text retrieval," in ASE, 2014, pp. 115–120.
[23] J. Sillito, F. Maurer, S. M. Nasehi, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in ICSM, 2012, pp. 25–34.
[24] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in MSR, 2014, pp. 112–121.
[25] P. S. Kochhar, "Mining testing questions on stack overflow," in 5th International Workshop on Software Mining, 2016, 2016, pp. 32–38.
[26] A. S. Badashian, A. Esteki, A. Gholipour, H. Abram, and E. Stroulia, "Involvement, contribution and influence in github and stack overflow," in CASCON, 2014, pp. 19–33.
[27] B. Gregoire, Y. He, and H. Alani, "A question of complexity: measuring the maturity of online enquiry communities," in 24th ACM Conference on Hypertext and Social Media, 2013, pp. 1–10.
[28] I. Srba and B. Maria, "A comprehensive survey and classification of approaches for community question answering," in ACM Trans. on the Web (TWEB), vol. 10, no. 3, Aug. 2016, pp. 18:1–18:63.
[29] E. Lehmann and J. Romano, Testing Statistical Hypotheses, ser. Springer Texts in Statistics. Springer New York, 2008.
[30] C. Treude, O. Barzilay, and M. A. Storey, "How do programmers ask and answer questions on the web?" in ICSE NIER, 2011, pp. 804–807.
[31] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack overflow," in ICSME, 2014, pp. 531–535.
[32] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. Boston, MA, USA: Addison-Wesley Longman Publ. Co., Inc., 2005.
[33] Laravel. (2017) Laravel. https://laravel.com/docs/installation.
[34] (2017) Node.js official docker image website. https://hub.docker.com/_/node/.
[35] (2017) Debian. http://www.debian.org/.
[36] G. user. (2017) Tar problem when installing meteor. https://github.com/meteor/meteor/issues/5762.
[37] ——. (2017) Automated build fails on 'tar' with: "directory renamed before its status could be extracted". https://github.com/docker/hub-feedback/issues/727.
[38] (2019) JSFiddle. https://jsfiddle.net.
[39] (2019) http://http://frisk.igorwiese.com/tutorial.
[40] (2019) Jade Language: Node Template Engine. .
[41] M. Liljedhal and J. Leibiusky. (2019) Play with docker. https://github.com/play-with-docker/play-with-docker.
[42] (2019) Play with docker labs. https://labs.play-with-docker.com/.
[43] (2019) Web framework rankings. https://hotframeworks.com/.
[44] M. Lorr, Cluster analysis for social scientists. Jossey-Bass.
[45] L. Melo, I. Wiese, and M. d'Amorin, "Dataset," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.3540911
[46] E. Horton and C. Parnin, "Gistable: Evaluating the executability of python code snippets on github," in ICSME, 2018.
[47] (2017) repl.it. https://repl.it.
[48] (2017) Cloud9. https://c9.io.
[49] (2017) Codeanywhere. https://codeanywhere.com/.
[50] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing stack overflow questions by topic, type, and code," in MSR, 2013, pp. 53–56.
[51] Apache. (2017) Lucene. https://lucene.apache.org/core/.
[52] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack overflow in github: Any snippets there?" in MSR, 2017, pp. 280–290.
[53] E. Horton and C. Parnin, "Dockerizeme: Automatic inference of environment dependencies for python code snippets," in ICSE, 2019, pp. 328–338.
[54] F. Hassan, R. Rodriguez, and X. Wang, "Rudsea: Recommending updates of dockerfiles via software environment analysis," in ASE, 2018, pp. 796–801.
[55] StackOverflow. (2017) Stackoverflow hiring trends 2017. https://stackoverflow.blog/2017/03/09/developer-hiring-trends-2017/.
[56] (2017) Stack-overflow. https://stackoverflow.com/insights/survey/2017.

Luis Melo is a Software Engineer at C.E.S.A.R., Brazil. He obtained his masters degree in Computer Science from the Federal University of Pernambuco (UFPE), Brazil. His current research interests include software security, testing and quality. More information is available at http://www.lhsm.com.br.

Igor Wiese is an Associate Professor in the Department of Computing at the Federal University of Technology – Parana, Brazil, where he is interested in Mining Software Repositories, Human Aspects of Software Engineering, and related topics. Wiese received a PhD degree in computer science from the University of São Paulo. More information is available at http://www.igorwiese.com.

Marcelo d'Amorim is an Associate Professor at the Federal University of Pernambuco (UFPE), Brazil. He obtained his PhD degree in Computer Science from the University of Illinois Urbana-Champaign (USA) in 2007. He is interested in preventing, finding, diagnosing, and repairing software bugs and vulnerabilities to improve software quality and productivity. More information is available at http://www.cin.ufpe.br/~damorim/.