

A Study of Update Request Comments in Stack Overflow Answer Posts

Mohammad Sadegh Sheikhaei^a, Yuan Tian^a, Shaowei Wang^b

^a*School of Computing, Queen's University, Kingston, ON, Canada*

^b*Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada*

Abstract

Comments play an important role in updating Stack Overflow (SO) posts. They are used to point out a problem (e.g., obsolete answer and buggy code) in a SO answer or ask for more details about a proposed answer. We refer to this type of comment as update request comments (URCs), which may trigger an update to the answer post and thus improve its quality.

In this study, we manually analyze a set of 384 sampled SO answer posts and their associated 1,221 comments to investigate the prevalence of URCs and how URCs are addressed. We find that around half of the analyzed comments are URCs. While 55.3% of URCs are addressed within 24 hours, 36.5% of URCs remain unaddressed after a year. Moreover, we find that the current community-vote mechanism could not differentiate URCs from non-URCs. Thus many URCs might not be aware by users who can address the issue or improve the answer quality. As a first step to enhance the awareness of URCs and support future research on URCs, we investigate the feasibility of URC detection by proposing a set of features extracted from different aspects of SO comments and using them to build supervised classifiers that can automatically identify URCs. Our experiments on 377 and 289 comments posted on answers to JavaScript and Python questions show that the proposed URC classifier can achieve an accuracy of 90% and an AUC of 0.96, on average.

Keywords: Stack Overflow, Answer quality, Crowd-sourced knowledge sharing, Commenting, Knowledge maintenance and update, Classification

1. Introduction

Many developers utilize Stack Overflow (SO) to find solutions for their programming issues. With about 22 million questions and 33 million answers¹, SO is the largest Q&A site for computer programming (May et al., 2019). According to Jeff Atwood, one of the founders of SO, the goal of SO is not “answer my question” but “let’s collaboratively build an artifact that will benefit future coders” (Atwood, 2018). As a result, most of the answers (sometimes the questions) on SO are required to be continuously edited to maintain/improve their quality via resolving the textual/code issues (e.g., handle deprecated APIs and fix flawed code snippet) in the previous version (Zhang et al., 2019; Wang et al., 2018a).

Comments on the SO posts are the main channel for users to communicate and discuss the potential problems in the posts. SO encourages users to post comments in an answer post when they find an issue in the answer and ask for an update on the answer explicitly (e.g., “Please replace method A with method B as A is deprecated.”) or implicitly (e.g., “So when using `ArrayList::new` the given key is inserted into the list?”). We refer to such comments as *update request comments (URCs)* because they can potentially trigger an update in the corresponding answer and thus improve its quality.

The questions and issues mentioned in URCs may be addressed in the next comment(s) or body of the corresponding answer post, or even both of them. However, there is no guarantee for such URCs to be addressed. In SO, when a user writes a comment under an answer post, the system notifies the owner of the post, i.e., *answer owner*, about the new comment. Then for each URC, the answer owner can address it either by updating the answer body or by writing a new comment to reply. However, if the answer owner does not handle the problem, the URC remains unaddressed until other users address it in a new comment or in the body of the answer (i.e., becoming an *answer editor*). Prior studies find only a small portion of their collected comments resulted in an update in the corresponding answer post, and answer owners ignore many requests of answer update raised in comments (Soni and Nadi, 2019; Zhang et al., 2019). In other words, handling such URCs is still problematic, and relying solely on answer owners to maintain their answers maybe not be realistic. SO is a collaborative community and all users on SO are encouraged to maintain its answers via collaborative editing (Wang

¹<https://stackexchange.com/sites>

et al., 2018a). Therefore, SO probably needs to attract eyeballs from the entire community to handle URCs. To alleviate the above problem, a first step is to have a deep understanding of URCs and investigate the possibility of developing an automated approach to identify such URCs so that they are visible to the community.

In this work, we first conduct an empirical study on URCs for having a deep understanding of URCs in terms of their prevalence, the percentages of URCs remained unaddressed, how URCs are addressed by the community members (i.e., addressed in answer post, addressed in the following comment(s), or addressed in both), how fast are URCs addressed, what is the contribution of different user roles in addressing URCs, which post part they choose to address URCs, and if comment votes can be used to distinguish URCs from non-URCs.

To answer the above questions, we manually examined 1,221 comments from a statistical randomly sampled 384 answer posts of Java questions (questions with the <java> tag). We observed that half of the analyzed comments are URCs. More interestingly, while 55.3% of URCs are addressed within 24 hours, 36.5% of URCs remain unaddressed after a year. We also found that the majority (80.1%) of addressed URCs are addressed by the answer owner. Among addressed URCs, 88.7% were addressed in the next comments, 33.3% were addressed in the post body, and 22.1% were addressed in both. By investigating the comment scores we realized that their scores are not good means to detect URCs.

Our findings show that although URCs are prevalent and more than half addressed within 24 hours, many are ignored (remained unaddressed). Moreover, these URCs might not be visible to the community as they may not be highly voted. Therefore, we continue to explore the feasibility of automated URC detection, as the first step towards improving the awareness of URCs. We propose multi-dimension features, such as comment author role and comment relative time, that could potentially differentiate URCs from other comments based on our manual analysis of 1,221 comments. Then we employ these features in common supervised learning models to identify URCs. Specifically, we apply random forest (Breiman, 2001), logistic regression (Hosmer Jr et al., 2013), naive bayes (Zhang et al., 2011), and also two deep learning models (a CNN model by Qu et al. (2019) and a Transformer based model by Gu and Budhkar (2021)), and train them by different inputs, i.e., the extracted features, TF-IDF/text, or extracted features + TF-IDF/text. We use the 1,221 annotated comments related to Java topic

(*Java comments*) to train these models, and then test them on comments extracted from JavaScript questions and Python questions to evaluate their performance and generalizability.

Our experiments on 377 and 289 comments posted on answers of JavaScript and Python questions respectively show that the models that are based on the extracted features outperform TF-IDF and text based models with a large margin in terms of accuracy and AUC. Also, among the investigated models, the Transformer based model (using BERT) that gets text and the extracted features as its input, achieves the highest performance, i.e., around 90% accuracy and 0.96 AUC, indicating that URCs are highly predictable.

Our contributions include:

- Perform an empirical study on the update request comments in Stack Overflow.
- Propose a supervised-learning approach that leverages our extracted features to detect URCs with the average accuracy of 90%. This approach can be used in Stack Overflow to decrease the rate of unaddressed URCs.
- Provide an annotated dataset of 1,221 comments² that are posted on randomly selected 384 answers to Java questions. We have provided three level of annotation: 1—If it is a URC, 2—for URCs, where is it addressed, i.e., in the next comments, in the post body, or both. 3—for URCs that are addressed by the following comments, which comment addresses it.

2. Background

As mentioned earlier, SO posts (either questions or answers) are continuously updated to address different issues such as resolving bugs, meeting time-related concerns (such as deprecated APIs), and simplification. According to Stack Overflow³ “*Any user can propose edits, but not all edits are publicly visible immediately. If a user has less than 2,000 reputation, the suggested edit is placed in a review queue. Two accept or reject votes are required to remove the suggested edit from the queue and either apply the edit*

²available at <https://doi.org/10.6084/m9.figshare.19382156>

³<https://stackoverflow.com/help/editing>

to the post or discard it. Users with more than 2,000 reputation are considered trusted community members and can edit posts without going through the review process.” As a result, if there is a problem with a post, users usually write a comment on the post and ask for an update. Also, users may ask for clarification or question other related cases, resulting in the generalization of a solution.

Fig. 1 shows a sample answer post from a question having the <java> tag along with its comments. There are four comments on this sample⁴. The first comment is written by the questioner (the user who started this page by posting the question) to ask more information about the answer. The second comment is written by the answerer (the user who wrote the answering post), to answer the first comment. The third comment is written by a third person to ask for more clarification, and the last comment is again from the answerer to address the third comment. Therefore, in this example, there are two URCs and two non-URCs. This sample also shows that there is an update on the answering post on Mar 29, 2016. By clicking on this date, SO opens a new page and shows the history of edits on this post by highlighting the new added parts in green and deleted parts in red. In this example, the last line of the answering post, i.e., “*For a list of implementations, including validators in various languages, see JSON-Schema Implementations.*”, is appended by the answerer to address the first comment. Therefore, as shown in the figure, the first comment is a URC addressed in the next comment and the post body. But, the third comment is a URC that is only addressed by the next comment. Both of these URCs are addressed in less than 24 hours. Fig. 2 shows another example⁵ from the Java community. Although the answer is accepted, four unaddressed update request comments point out the problems with the proposed answer.

The effect of comments on answer updates was studied before by Soni and Nadi (2019). They employed three rule based heuristics to detect different types of comments regarding their effect on posts, although that approach discards around 35% of comments due to not matching to some primary conditions. In contrast, we created a dataset of 1221 comments by providing three levels of annotation for each comment. We employed this dataset to (A) answer different empirical questions about URCs, (B) train various ML

⁴<https://stackoverflow.com/questions/36152972>

⁵<https://stackoverflow.com/questions/27304654>

Validate JSON against XML Schema (XSD)

No, [XML Schema \(XSD\)](#) is for validating [XML](#); to validate [JSON](#), see [JSON Schema](#).

I recommend generating schemas by hand for full understanding and full control over the constraints. However, here are some automated tools that can jumpstart the process:

- To convert from JSON Schema to XSD, see [jsons2xsd](#).
- To convert from XSD to JSON Schema, see [Jsonix Schema Compiler](#).

Related and also very useful:

- To parse from XML to JSON (unmarshal) or serialize JSON to XML (marshal), see [JSONIX](#).
- For a list of implementations, including validators in various languages, see [JSON-Schema Implementations](#).

Share Follow edited Mar 29 '16 at 10:58 answered Mar 22 '16 at 13:12 kj 92.6k ● 17 ● 147 ● 205

URC Thank you. Is it easy to validate against the JSON schema? – us Mar 29 '16 at 8:49

NO_URC Generally, sure. See [Validators](#) section of [JSON-Schema Implementations](#). – kj Mar 29 '16 at 10:55

URC Can you expand on why you can't use XSD for JSON? I imagine you could just convert the JSON to XML then check that XML against the XSD. Other than raw text in XML, JSON and XML are just elements which have attributes and child elements. – ub Jun 7 '19 at 21:09

NO_URC @ub: Of course any XML can be validated with XSD, including XML that was automatically converted from JSON using the methods I mentioned in the answer. – kj Jun 8 '19 at 2:27

Figure 1: A sample answer post from a question tagged with “Java”. There are two addressed update request comments and two non-update request comments on this answer.

and DL models to predict the type of comments. Due to the differences between our approach and Soni’s work, we come to a significantly different conclusion on the ratio of addressed and unaddressed URCs. The details have been presented in Section 7.

3. Empirical Analysis on Update Request Comments in SO Answers

To understand the prevalence of update request comments (URCs) and how they are handled in practice, we first conduct an empirical study by answering the following four research questions. Answers to them would also guide future studies leveraging SO comments. As far as we know, this is the first empirical analysis of URCs.

RQ1: How prevalent are URCs in technical Q&A and how they

How to find the smallest distance between two numbers next to each others in an array?



-2



```
int smallest = 100000
for (int i = 0; i < array.length; i++)
    if(array[i+1]-array[i]<smallest)
    {
        smallest = array[i+1]-array[i];
    }
System.out.println(smallest);
```

Share Improve this answer Follow

answered Dec 4 '14 at 21:41



60 ● 7

1 What about if the smallest is 100001? – D [REDACTED] Dec 4 '14 at 21:44

1 should probably stop at array.length-1 to keep from going out of bounds. – Sei [REDACTED] Dec 4 '14 at 21:44

2 I accepted your answer not because it worked 100%, rather you helped me to clear the concept. However array.length reaches out of the array, thus, I modified it to array.length-1. And I defined the smallest integer to Integer.MAX_value. Thanks for helping me, xoxo. – Si [REDACTED] Dec 4 '14 at 22:39

@Stl [REDACTED] Its better you do the corrections in the answer as told by Simon in the comments – n [REDACTED] Jul 8 '17 at 3:11

Figure 2: A sample answer posts from a question tagged with “Java”. There are four unaddressed update request comments on this answer.

get addressed by the community members? Knowing the percentage of URCs and the number of unaddressed URCs among all comments gives us insight into how critical it is to analyze URCs. Other information such as the ratio of URCs caused a post update gives us a higher level of perception about the user interactions via comments to improve the quality of answers.

RQ2: How fast are URCs addressed? Knowing the URCs address speed gives us insight into the delay in addressing URCs, i.e., potential waiting time for SO users to get their URCs addressed. This metric could also help the SO community to keep track of the effectiveness of the community in addressing URCs.

RQ3: Which user role (questioner, answerer, other commenters) is more likely to address URCs? And in which part of the answer post do they choose to address URCs? Users who address URCs may have different roles in that SO page: questioner, answer owner, answer editor,

and others. Also, they may address URCs in the post body, in the following comments, or both. Knowing which user role addressed URCs in which part of the answer post gives us insight on who contributes most to the address of URCs and where they prefer to address URCs.

RQ4: Can comment votes be used to distinguish URCs from non-URCs? SO uses a community-vote mechanism to decide which comments should be shown at top positions. Knowing if votes can help identify URCs gives us insight into whether URCs can be naturally selected by SO users or not.

To answer the above questions, we collect a set of comments and determine whether each of them is a URC or not. For each URC, we also denote if the URC is addressed or not. For addressed URC, we record if they are addressed in the answer post or in the following comments, and identify the role of SO user who addressed the URC. Section 3.1 and 3.2 describe the coding guide for comment labeling and how we collect data for answering four RQs. In the end, we present the methodology and results of our empirical analysis in Section 3.3.

3.1. Coding Guide to Annotate the Comments

Before labeling the comments, we need a coding guide to annotate the comments by URC/NO.URC, and then labeling the URC comments by URC_ADDRESSED/ URC_UNADDRESSED. A URC is a comment posted by any user but the answerer, explicitly or implicitly, asks to update the answer and improve its quality. As the answerer has the ability of changing his/her answering post, comments by this person is a NO.URC comment. For the other comments that are from the questioner or third users, we check the content of the comment. If it either points out problems in the answer post, asks for more information that would help to understand the answer better, or provides important information (e.g., regulations) associated with the answer, we label it as URC (because it has the potential of initiating a post update), otherwise we label it as NO.URC. Fig. 3 shows this process in a decision tree. According to this figure, the first and third comments in Fig. 1 are URC because they ask for more information about a special case that would help the questioner to better understand the answer. However, the second and fourth comments are NO.URC because they are written by the answerer. To provide more examples, these are some comments by a questioner or a third person on an answer post that we consider them as

URC:

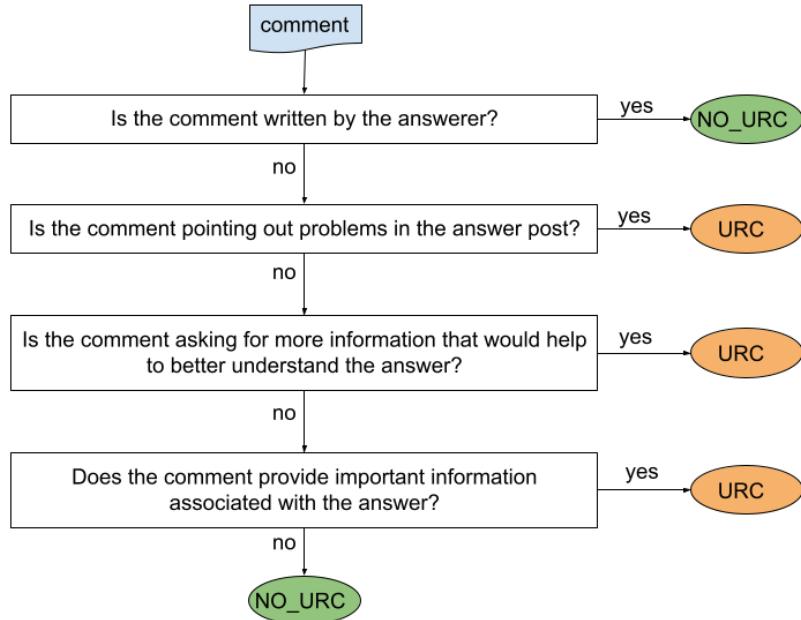


Figure 3: The decision tree applied in manually labeling comments by URC/NO_URC

- How can I use this solution in my code?
- I ran it and got this Exception: ...
- Can I use class B rather than class A in this solution?
- It doesn't work for me.
- Function A is deprecated.
- Works but runs slowly.

And here are examples of comments by a questioner or a third person on an answer post that we consider them as NO_URC:

- Thank you!
- Great! This is the thing I was looking for.
- Oh, that's a nice point about having the possibility to put a null value there!

To tag the URCs with URC_ADDRESSED/URC_UNADDRESSED label, we add another level of annotation which determines if the URC is addressed in the following comments, or addressed in the post body, or addressed in both of them, or remained unaddressed. Any answering comment related to the URC (even if it is not the correct answer) is acceptable to tag that comment as URC_ADDRESSED. We assume that if the answering comment is not what the user asked, she will write another URC. The only answering comments that we do not treat as answers are those that explicitly say “I don’t know” and so forth.

To answer RQ2 and RQ3, we need to know which one of the next comments addresses the current URC (if any). Thus, the final step of our annotation process is to add another level of labeling to determine the addressing commentID (when the URC is addressed by the next comments). If there are multiple addressing comments, we take the first (the oldest) of them. As a result, our dataset has three levels (columns) of labeling, i.e., needs_update, addressed_in, and addressed_by_commentID. Table 1 shows our labeling for the comments on the two answer posts mentioned in Fig. 1 and 2. Due to lack of space, only the first few words of each comment text is shown. In the “user role” column, we mentioned the role of the commenter. Refer to Table 6 for more information about these roles. The three levels of our annotations are presented in “needs update”, “addressed in”, and “addressed by commentID” columns.

Table 1: Labels for the comments of two answer posts from SO Java community

question ID	answer ID	comment ID	user role	Comment text	needs update	addressed in	addressed by comment ID
36152972	36155326	60185364	Questioner	Thank you. Is it easy to ...	yes	both	60190443
36152972	36155326	60190443	Answerer	Generally, sure. See ...	no	-	-
36152972	36155326	99591573	Not seen commenter	Can you expand on why ...	yes	comment	99594548
36152972	36155326	99594548	Answerer	@ubiquibacon: Of course any ...	no	-	-
27304556	27304654	43072230	Not seen commenter	What about if the smallest ...	yes	no	-
27304556	27304654	43072237	Not seen commenter	should probably stop at ...	yes	no	-
27304556	27304654	43073707	Questioner	I accepted your answer not ...	yes	no	-
27304556	27304654	76939452	Not seen commenter	@StevenAkaTaz Its better ...	yes	no	-

3.2. Data Collection

We collect data from the 2020_03_15 version of SOTorrent dataset (Baltes et al., 2019) by performing queries on Posts, PostHistory, Users, and Comments tables.

To achieve more reliable conclusions (less diverse results), in this study we focus on SO posts that are related to Java language (have `<java>` tag). We expect to see similar results on other popular programming languages such as Python and JavaScript. We also consider only the questions that their score is equal or greater than zero. We assume that negative scored questions/answers have not attracted enough interest from the community to evaluate and classify their quality. As we aim to investigate the comments on answer posts, only the answers that have at least one comment are considered. Given the huge number of candidate answer posts, we focus on those that are recently touched, i.e., the last activity date (last post edit date, or post creation date if it is not edited) is 1/1/2017 or later. Moreover, the accepted answers or the answers that have the highest vote among answers posted on the same question are more important for the community. So, we only consider answers that either is accepted or has the highest vote. After filtering the answers based on the mentioned features, we ended up with 124,472 answers. Then, we randomly choose a sample of 384 answers to fulfill 5% margin error with a confidence level of 95% for our statistical analysis. To calculate the sample size, we apply the sample size calculator⁶. Finally, we take all the comments of that 384 posts for our annotation process. Algorithm 1 shows the steps of data acquisition from SOTorrent.

Algorithm 1 Data Acquisition from SOTorrent

- 1- Select all questions with Java tag and $Score \geq 0$
 - 2- Select answers of questions selected in Step 1 that A) is accepted or has the highest votes among answers posted on that question, B) posted or edited after 1/1/2017, and C) have one or more comments
 - 3- Randomly select 384 answers from the set of answers made in step 2
 - 4- Select all comments of the 384 answers selected in the step 3
-

To manually label the data, we follow the closed card sorting methodology. According to the coding guide for labeling the comments introduced in Section 3.1, the first two authors manually examined 100 randomly sampled comments obtained from step 4, and only found six inconsistent annotations. They discussed and agreed on the final labels of the six comments. The first author then continues to label the rest comments. When annotating the comments, we found two answers with ambiguous comments. We ignored

⁶<http://www.raosoft.com/samplesize.html>

those two answers, and ended up with 1,221 annotated comments. Using that 100 randomly sampled comments that were annotated independently by the first two authors, the Cohen's kappa inter-rater agreement is 87.8%, which indicates an excellent agreement. We will publish our dataset online for future research around this subject.

3.3. Methodology and Results

RQ1: How prevalent are URCs in technical Q&A and how they get addressed by the community members?

We use our labeled dataset to answer this question. Table 2 shows the main statistical questions about update request comments. About half of the comments (631 of 1,221) are URC. Among 631 URCs, 417 comments (about 66%) are addressed either by a post update or by another comment, and about 34% of URCs remained unaddressed in our dataset. A strong majority (88.7%) of the 417 addressed URCs are addressed in the next comment. Also, 139 comments (33.3%) of these addressed URCs are addressed in the post body while 92 of them are addressed in the next comments as well.

Table 2: The main statistics about URCs in our dataset

Statistical Questions	Count	Percent
How many comments are URC?	631 of 1221	51.7%
How many of URCs are addressed (either in post or next comments)?	417 of 631	66.1%
How many of the addressed URCs are addressed in the next comments?	370 of 417	88.7%
How many of the addressed URCs are addressed in the post body?	139 of 417	33.3%
How many of the addressed URCs are addressed both in the next comments and the post body?	92 of 417	22.1%

RQ2: How fast are URCs addressed?

Same as RQ1, we use the annotated dataset to answer this question. In our labeling, we didn't mention which post-update addresses the URC. But, we have labeled whether each URC is addressed by post-update or not. Thus, in case of being addressed in the post body, we assume the first post-update after that URC is the one that addresses it.

Table 3 shows the percentage of URCs that are addressed within 5 minutes, 1 hour, 1 day, 7 days, and a year. The table reports the portions based on both the 417 addressed URCs and all 631 URCs. Interestingly, 20% of 631 URCs are addressed within 5 minutes, and about 55.3% of them are addressed within a day. Among 417 addressed URC, 400 comments (95.9%)

were addressed within a year, meaning that 17 comments waited for more than a year to get addressed. On the other hand, among 47 unaddressed URC that were posted after 3/15/2019, one of them addressed within a year but after 3/15/2020. So, among 631 URCs, $417-17+1=401$ comments (63.5%) addressed within a year, and 230 comments (36.5%) remained unaddressed within this period.

Table 3: The percentage of addressed URCs within the specified times

Addressed within	Of 417 addressed URCs	Of 631 URCs
5 min	30.2%	20.0%
1 hour	58.8%	38.8%
1 day	83.7%	55.3%
7 days	87.5%	57.8%
1 year	95.9%	63.4%

RQ3: Which user role (questioner, answerer, other commenters) is more likely to address URCs? And in which part of the answer post do they choose to address URCs?

Same as previous RQs, we investigate our dataset to answer this question. Table 4 shows the results. The rows show who (which user role) addressed the URCs, and the columns show where (which part of the post) URCs are addressed. There are 370 URCs that are addressed in the next comments, which a majority of them (296) are addressed by the answer owner (i.e., the person who posted the answer for the first time). The remaining are addressed by answer editor, i.e., the user who update the existing answer (3 URCs), by the questioner (15 URCs), and the other users (56 URCs).

Table 4: Role of users who address the URCs and where they address the URCs

URCs addressed	in comment	in post	in either	in both
by answer owner	296	114	334	76
by answer editor	3	25	27	1
by questioner	15	0	15	0
by others	56	0	56	0
by anyone	370	139	417	92

There are 139 post edits (to address URCs) which 114 of them are by the post owners and the remaining 25 post edits are by answer editors.

The answer owners addressed 334 URCs either in the post or in the next comments. It means that $334/417=80.1\%$ of addressed URCs are addressed by the answer owners. The table also shows that 76 URCs are addressed by the post owner both in the next comments and in the post body.

Note that the values in “by anyone” row for “in either” and “in both” columns are not the summation of that columns. For example, among 92 URCs that are addressed both in the next comments and in the post body, some of them are addressed by post owner in the post body (without writing a comment by them) while they also addressed by others in the comment.

RQ4: Can comment votes be used to distinguish URCs from non-URCs?

Table 5 shows the different quantiles of comment scores (votes) for each group. The comment score for more than 75% of comments of each group is 0. Also, the quantiles for 80% and 85% are equal, indicating that the comment scores are not good means to detect URCs.

Table 5: The quantiles of comment scores for each group of comments

Category	50%	75%	80%	85%	90%	95%
NO_URC	0	0	1	1	1	2
URC	0	0	1	1	2	4

Summary: About half of the comments are URC. While 55.3% of URCs are addressed within 24 hours, 36.5% of URCs remain unaddressed after a year. Also, the majority (80.1%) of addressed URCs are addressed by the answer owner. The majority URCs’ score is 0, which may not be visible to the community.

4. Automatically Detect Update Request Comments

4.1. URC Detection

We know that SO notifies the owner of posts when someone writes a new comment on their post. This SO feature explains why most of the URCs (80.1%) are addressed by the owner of the answering post, and why 55.3% of the URCs are addressed within 24 hours, whereas 36.5% remain unaddressed after a year. In addition, we observe that the majority URCs have a score of 0, which may cause them invisible to community members

who are potentially interested in addressing them. Therefore, if there is a reliable predictive model to automatically detect URCs, Stack Overflow can apply it to improve post qualities by decreasing the number of unaddressed URCs. For example, when a new URC is posted on an answer and accurately identified by a URC detector, if the answer remains unchanged and no other comments appear after the identified URC, SO could predict this URC to be an unaddressed URC and push it to the community members who are interested in addressing the issue pointed out in the URCs, including the original answer. Thus, in this section we aim to investigate how URCs are predictable.

To predict URCs, we apply three classic ML models and two deep learning models (a CNN and a BERT based model). Each ML model utilizes either the features extracted from comments, or the TF-IDF of the comment text or both of them (i.e., the extracted features and the TF-IDF table that are horizontally concatenated). Each DL model utilizes either the text only or the text plus the features extracted from comments. We describe the details of the feature extraction process, TF-IDF extraction, and training classifiers as below.

Feature Extraction: Table 6 describes the features we extracted for each comment by considering multiple dimensions, inspired by our manual comment annotation process. These dimensions are as follow:

- **Comment features:** the features that are related to global aspects of a comment. It includes comment_score and comment_order.
- **Post features:** the features that are related to global aspects of a post. It includes post_score and post_comment_count.
- **User features:** we believe the role of the user who posts a comment is an important clue to detect URCs. Most NO_UCRs are posted by the answerer to address previous URCs. We also consider user_reputation that is provided by Stack Overflow. As users with high reputation have the permission of updating answers, they may update answers by themselves in case of new URCs, and write new comments when they want to address a URC.
- **Time features:** These features are the relative time between a comment and its previous/next comment and its previous/next post up-

dates. The rational behind these features is that, in most cases URCs are addressed within a short period of time.

- **Text similarity features:** Each comment is more or less related to other comments and post updates. In most cases, they are related to immediately previous/next comment or the immediate next post update. We use two different text similarity measures, i.e., the Jaccard similarity and cosine similarity between the BERT vectors (obtained via running SBERT⁷ on each comment) to extract the similarities between a comment and its previous/next comments. We also use the Jaccard similarity to find the similarity between a comment and the next post change occurred after that comment. So, by extracting these similarities, we will find that how near events are related to the current comment.
- **Text semantic features:** We use TextBlob (Loria, 2018), a Python text processing library to extract the polarity and subjectivity of comments. We expect critic comments (that are kind of URCs) have a negative polarity.
- **Text extracted features:** There are some characters such as question mark, keywords such as “exception” or “but”, URLs, and emotions (like ;)) that knowing their existence in a text provides good information about the content of a comment. We also considered the text length, because it somehow shows how much information is provided by that comment.

TF-IDF Extraction: Term frequency-inverse document frequency (TF-IDF) is one of the most popular methods to vectorize documents of a corpus. A word in the TF-IDF of a document achieves a high value if it has a high frequency in that document but a low frequency in the whole collection of documents. To archive the best results, we remove the stop words and also the words with very low frequency, i.e., one or two.

Training Classifiers: We consider three basic classification models, i.e., random forest, logistic regression, and naive bayes. These models are picked as they are most commonly used in prediction tasks in Software Engineering

⁷<https://www.sbert.net/>

Table 6: The extracted features from each comment

Feature	Description
Comment Features:	
comment_score	The score of the comment which is zero or a positive number.
comment_order	The order of the comment on that post. The first comment is 1, the second is 2, and so on.
Post Features:	
post_score	The score of the post that can be a positive or negative number
post_comment_count	The number of comments on that post.
User Features:	
by_asker	True: if the comment is written by the user who posted the question. False: otherwise.
by_answerer	True: if the comment is written by the user who posted the answer. False: otherwise.
by_not_seen_commenter	True: if the comment is written by a user that is neither the questioner nor the answerer, and has not written any comment on this post before. False: otherwise.
by_seen_commenter	True: if the comment is written by a user that is neither the questioner nor the answerer, but has written at least one comment on this post before. False: otherwise.
user_reputation	The reputation of the user who wrote the comment
Time Features:	
prev_post_edit_time	$\log([\text{The time of the comment}] - [\text{The time of the previous post edit}])$ reported in minutes
next_post_edit_time	$\log([\text{The time of the next post edit}] - [\text{The time of the comment}])$ reported in minutes
prev_comment_time	$\log([\text{The time of this comment}] - [\text{The time of the previous comment}])$ reported in minutes
next_comment_time	$\log([\text{The time of the next comment}] - [\text{The time of this comment}])$ reported in minutes
Text Similarity Features:	
prev_comment_jaccard_sim	The Jaccard similarity between this comment and the previous comment
next_comment_jaccard_sim	The Jaccard similarity between this comment and the next comment
prev_comment_bert_sim	The Cosine similarity between the BERT vector of this comment and the previous comment
next_comment_bert_sim	The Cosine similarity between the BERT vector of this comment and the next comment
comment_post_change_sim	The Jaccard similarity between this comment and the post change after this comment
Text Semantic Features:	
polarity	The polarity of the comment text (between -1 and +1)
subjectivity	The subjectivity of the comment text (between 0 and 1)
Text Extracted Features:	
text_len	The length (number of characters) of the comment text.
starts_with_@	If the comment starts with @. For example: “@John please explain your code.”
contains_question_mark	If the comment contains a question (?) mark.
contains_exclamation_mark	If the comment contains an exclamation (!) mark.
contains_but	If the comment contains the word “but”.
contains_exception	If the comment contains the word “exception”.
contains_url	If the comment contains a URL.
contains_emotions	If the comment contains emotions like :)
talks_to_role	If this comment talks to a specific person by @. 0: doesn’t include @user, 1: talks to the questioner, 2: talks to the answerer, 3: talks to a commenter

domain (Yang et al., 2020), and have shown acceptable performance in those tasks. We also consider two deep learning models:

1) a convolutional neural network (CNN) proposed by Qu et al. (2019) that has the capability of incorporating the external features to the neural network (after the convolutional layers). To get the most from the CNN model, we modified its layers and tune its hyper parameters according to the 10% validation set that is randomly selected from 1,221 java comments. In the original implementation of the CNN model, the authors used three convolutional layers. Also they directly concatenated the external features to the convoluted text. However, we found that if we decrease the number of con-

volutional layers from three to one, and also if we use a dense layer before concatenating the external features to the convoluted text we get better results. For the hyper parameters, we used the same optimizer (Adam) and same parameter values except for the learning rate that was 0.001 but we changed it to 0.0005.

2) Multimodal-Toolkit, the deep learning model proposed by Gu and Budhkar (2021) that gets text and tabular data and provides eight different architectures to combine the tabular data with a Transformer (such as BERT). We use bert-base-uncased as the Transformer in this toolkit. Our primitive investigations on that eight different architectures showed that we can achieve the highest performance by the architecture that embeds the text to a vector of size 768, then concatenate it with the output of a MLP which gets the tabular data (non-text data) and converts them to a vector of size 500. To be more specific, the MLP has one hidden layer with 10,000 nodes, and there are 500 nodes in its output layer. So, the concatenated vector has a size of $768+500=1268$. For the hyper parameters, we use the default parameters and suggested values. To be more specific, the batch size is 16, the Adam learning rate is 5e-5, and the number of training epochs is 3.

4.2. Experiments

Evaluation Metrics: The automated identification of URCs can be treated as a standard binary classification task. Thus, we use standard evaluation metrics, i.e., precision, recall, F1-score, accuracy, and area under the curve (AUC) to evaluate our models. Precision P measures the correctness of our models in predicting the type of a comment, i.e., whether the comment is a URC or not. A prediction is considered correct if the predicted type is the same as the actual type of the comment. Precision is calculated as the proportion of correctly predicted URCs. Recall R measures the completeness of a model. A model is considered complete if all of update request comments are predicted to be URC. Recall is calculated as the proportion of actual URCs that were correctly predicted as such. F1-score is the harmonic mean of precision and recall, i.e., $(\frac{2*P*R}{P+R})$. Accuracy is the most intuitive performance measure and it is the ratio of correct predictions to the total predictions. The area under a receiver operating characteristic (ROC) curve, abbreviated as AUC, measures the overall performance of a binary classifier (Hanley and McNeil, 1982). The AUC value is within the range [0.5–1.0], where 0.5 represents the performance of a random classifier and 1.0 corresponds to a perfect classifier.

Experiment Setup: We use the 1,221 labeled comments from the SO Java community to train the models. To test these models, we create two test datasets, one from SO JavaScript community, and another from SO Python community. There are three main reasons behind the cross-programming language evaluation setup. First, due to the time limit, similar to literature studies on SO posts, e.g., Soni and Nadi (2019) and Tang and Nadi (2021), we do not consider all questions on Stack Overflow but those tagged with specific programming languages. We pick Java, Python, and JavaScript, as they are reported to be among the most popular programming languages⁸. Second, it would be easier for us to label URCs in posts tagged with these programming languages, as we have enough domain knowledge to understand the background and content of the questions. Last but not least, we choose to perform cross-programming language prediction because URCs in different programming language communities might share different characteristics, and we would like to investigate if the predictive models trained from one programming language can perform stably in other programming languages. As all the extracted features shown in Table 6 are language independent, all the applied ML and DL models in our experiments are language independent consequently. Therefore, we can train the models on the Java community comments, and test them on different datasets from different domains (e.g., JavaScript or Python), and expect similar performance on each domain.

To create the test datasets, we tag 377 comments (posted on 100 randomly selected answers) from the SO JavaScript community and 289 comments (posted on 100 randomly selected answers) from the Python community. To randomly select 100 answering post from JavaScript or Python community, we follow the steps described in Algorithm 1, but using <javascript> or <python> tag.

As random forest is a stochastic algorithm, it provides different results in each run. So, we run this algorithm for 100 iterations and report the result with the median accuracy. In each iteration, we train the algorithm by the 1,221 comments from the Java community and test it on 377 comments from the JavaScript community and 289 comments from the Python community. For logistic regression and naive bayesian models, as they provide stable results, we only run them for once. As the CNN model provides stable results

⁸<https://survey.stackoverflow.co/2022/#technology>

through different runs, we also run it for once. For Multimodal-Toolkit, we run it for 11 times and take the result with median accuracy.

Among the features mentioned in Table 6, six features may not be available in real scenarios: `next_comment_jaccard_sim`, `next_comment_bert_sim`, `comment_post_change_jacc_sim`, `next_post_edit_time`, `comment_score`, and `next_comment_time`. These features need some time to be available when a new comment is posted. So, we drop them before running the experiments.

Baseline: We also compare our models with the heuristic approach provided by Soni and Nadi (2019) (details provided in Section 7). They use three heuristics that are based on regular expressions and the code parts that are common between comments and post updates.

The heuristic algorithm proposed by Soni and Nadi generates four labels, three of which are equivalent to our labels, but their UNKNOWN label is undefined in our labeling. Moreover, some comments are discarded by their algorithm. We decided to treat UNKNOWN and discarded comments in two different ways: One way is to treat all of them as NO_UCR. The second way is to drop all UNKNOWN or discarded comments and report the performance on the remaining comments. Among 377 JavaScript comments, 146 comments were discarded or labeled as UNKNOWN by their heuristic algorithm. For the Python community, among 289 comments, 101 comments were discarded or labeled as UNKNOWN by this heuristic.

4.3. Results

Table 7 and Table 8 show the performance of the three ML models (that each one is trained by three different inputs), the CNN model (trained with two different inputs), the Multimodal-Toolkit by Gu and Budhkar (2021) that uses BERT (trained with two different inputs), and two baselines adjusted from Soni and Nadi’s heuristic approach, on JavaScript and Python respectively. The results show that for both test data, the Multimodal-Toolkit trained by the extracted features + text achieves the best performance, i.e., about 90% accuracy and 0.96 AUC. The random forest model trained by features+TF-IDF archives the second rank in the list for both test data.

The results in Table 7 and Table 8 also reveal that when we only use the text data (TF-IDF or pure text), none of the models achieves an accuracy higher than 74%. However, when we only employ the extracted features, all of the three ML models achieve much higher performance comparing their TF-IDF based models. Also, for the deep learning based models (CNN and

Multimodal-Toolkit), incorporating the extracted features resulted in a much higher performance comparing to its text based version. As expected, among the models which only uses the text data, either pure text or TF-IDF, the BERT based models provide the highest performance.

The two considered baselines did not perform well on our datasets. One potential reason is that regular expressions may not be as accurate as ML approaches. Moreover, we included additional important features such as the role of commenters. Finally, their definition of URC is not exactly the same as ours. The details are presented in section 7.1.

Table 7: The performance of different models with different input features on JavaScript comments

Classifier (Input)	Acc AUC	Category	P	R	F1
RandomForest (features)	88.3%	NO_URC	0.886	0.867	0.876
	0.946	URC	0.881	0.898	0.889
RandomForest (TF-IDF)	60.7%	NO_URC	0.611	0.489	0.543
	0.655	URC	0.605	0.716	0.656
RandomForest (features + TF-IDF)	89.4%	NO_URC	0.902	0.872	0.887
	0.949	URC	0.887	0.914	0.900
LogisticRegression (features)	70.8%	NO_URC	0.765	0.561	0.647
	0.769	URC	0.678	0.843	0.751
LogisticRegression (TF-IDF)	65.0%	NO_URC	0.638	0.617	0.627
	0.686	URC	0.660	0.680	0.670
LogisticRegression (features + TF-IDF)	84.1%	NO_URC	0.895	0.756	0.819
	0.928	URC	0.804	0.919	0.858
GaussianNB (features)	66.3%	NO_URC	0.627	0.728	0.674
	0.730	URC	0.708	0.604	0.652
GaussianNB (TF-IDF)	55.4%	NO_URC	0.524	0.733	0.611
	0.600	URC	0.616	0.391	0.478
GaussianNB (features + TF-IDF)	66.3%	NO_URC	0.627	0.728	0.674
	0.730	URC	0.708	0.604	0.652
CNN (text)	66.8%	NO_URC	0.655	0.644	0.650
	0.706	URC	0.680	0.690	0.685
CNN (text + features)	88.3%	NO_URC	0.915	0.833	0.872
	0.947	URC	0.859	0.929	0.893
Multimodal-Toolkit by Gu and Budhkar (2021) using BERT (text)	74.0%	NO_URC	0.720	0.744	0.732
	0.820	URC	0.759	0.736	0.747
Multimodal-Toolkit by Gu and Budhkar (2021) using BERT (text + features)	89.9%	NO_URC	0.908	0.878	0.893
	0.953	URC	0.892	0.919	0.905
Heuristic by Soni and Nadi (2019) (treat UNKNOWN and discarded comments as NO_URC)	49.3%	NO_URC	0.480	0.744	0.584
		URC	0.531	0.264	0.353
Heuristic by Soni and Nadi (2019) (ignore UNKNOWN and discarded comments)	50.2%	NO_URC	0.481	0.582	0.527
		URC	0.531	0.430	0.475

Fig. 4 shows the feature importance obtained by the feature based RF model with the median accuracy on JavaScript comments. As expected,

Table 8: The performance of different models with different input features on Python comments

Classifier (Input)	Acc AUC	Category	P	R	F1
RandomForest (features)	87.2%	NO_URC	0.930	0.811	0.866
	0.943	URC	0.825	0.936	0.877
RandomForest (TF-IDF)	63.0%	NO_URC	0.672	0.541	0.599
	0.679	URC	0.600	0.723	0.656
RandomForest (features + TF-IDF)	87.9%	NO_URC	0.931	0.824	0.875
	0.947	URC	0.835	0.936	0.883
LogisticRegression (features)	72.3%	NO_URC	0.788	0.628	0.699
	0.786	URC	0.678	0.823	0.744
LogisticRegression (TF-IDF)	63.7%	NO_URC	0.684	0.541	0.604
	0.694	URC	0.605	0.738	0.665
LogisticRegression (features + TF-IDF)	83.4%	NO_URC	0.910	0.750	0.822
	0.933	URC	0.778	0.922	0.844
GaussianNB (features)	64.4%	NO_URC	0.615	0.811	0.700
	0.741	URC	0.702	0.468	0.562
GaussianNB (TF-IDF)	59.5%	NO_URC	0.578	0.777	0.663
	0.635	URC	0.633	0.404	0.494
GaussianNB (features + TF-IDF)	64.4%	NO_URC	0.615	0.811	0.700
	0.741	URC	0.702	0.468	0.562
CNN (text)	64.7%	NO_URC	0.669	0.615	0.641
	0.682	URC	0.627	0.681	0.653
CNN (text + features)	87.5%	NO_URC	0.889	0.865	0.877
	0.953	URC	0.862	0.887	0.874
Multimodal-Toolkit by Gu and Budhkar (2021) using BERT (text)	74.0%	NO_URC	0.735	0.770	0.752
	0.826	URC	0.746	0.709	0.727
Multimodal-Toolkit by Gu and Budhkar (2021) using BERT (text + features)	90.7%	NO_URC	0.923	0.892	0.907
	0.969	URC	0.890	0.922	0.906
Heuristic by Soni and Nadi (2019) (treat UNKNOWN and discarded comments as NO_URC)	50.5%	NO_URC	0.513	0.669	0.581
		URC	0.490	0.333	0.397
Heuristic by Soni and Nadi (2019) (ignore UNKNOWN and discarded comments)	50.5%	NO_URC	0.522	0.495	0.508
		URC	0.490	0.516	0.503

`by_answerer` has the highest weight because most of the addressing comments (that are NO_URC) are written by the post answerer. The feature importance for other running iterations of RF is similar to this figure.

Summary: The update request comments are highly detectable. Utilizing the features we proposed to extract from comments, the supervised models achieved 89.9% and 90.7% accuracy on JavaScript and Python community respectively.

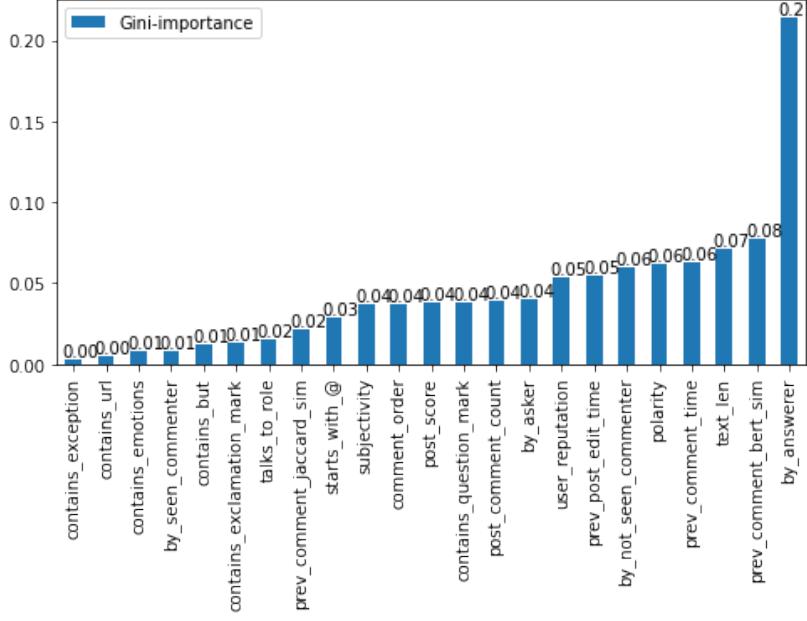


Figure 4: Feature importance by the random forest model

5. Discussion: Can we automatically identify unaddressed update request comments?

Identifying unaddressed URCs from existing SO comments might also help the SO community to improve the awareness of potential unresolved issues in existing answer posts. Thus in this discussion, we explore whether comment features proposed in Section 4 can also be applied to identify unaddressed URCs in existing SO comments.

Features and Model: We reuse the features proposed in Table 6 (including the six not available to new comments) and apply the Multimodal-Toolkit with BERT (as the best model to identify URCs) to detect all three classes: NO.URC, URC_ADDRESSED, and URC_UNADDRESSED. We train the model on the 1,221 Java comments and test it on 377 and 289 comments from accepted answers to JavaScript and Python questions, respectively.

Baseline: We compare our results with the heuristic rule-based model provided by Soni and Nadi (2019) (ref. Section 7).

Experiments and Evaluation: Following the same method of performance evaluation which is described in Section 4.2, we run the Multimodal-Toolkit for 11 times and report median accuracy value.

Results: Table 9 shows the performance measures for each category for both JavaScript and Python communities. The accuracy on JavaScript and Python comments is 84.1% and 84.8% respectively. However, the model cannot provide a high f1-score for the URC_UNADDRESSED class indicating the difficulty of detecting this type of comments.

Table 9: The performance of Multimodal-Toolkit using BERT to detect 3-class comments in JavaScript and Python communities

Community	Category	P	R	F1	Supp.
JavaScript (Acc: 84.1%)	NO_URC	0.869	0.922	0.895	180
	URC_ADDRESSED	0.868	0.797	0.831	148
	URC_UNADDRESSED	0.660	0.673	0.667	49
Python (Acc: 84.8%)	NO_URC	0.905	0.905	0.905	148
	URC_ADDRESSED	0.827	0.910	0.867	100
	URC_UNADDRESSED	0.645	0.488	0.556	41

Table 10 reports the performance of the heuristic model on JavaScript and Python comments when we treat the discarded and UNKNOWN comments as NO_URC. It provides 39.8% and 40.1% accuracy on JavaScript and Python comments respectively that is far lower than the performance of our proposed model. As treating the discarded and UNKNOWN comments by the second way (i.e., ignoring them) led to even worse performance (i.e., 34.6% and 34.6%), we don't present the detailed performance obtained by this treatment. The above results show that our model outperforms the baseline with a large margin in terms of accuracy and f1-score.

Table 10: The performance of the heuristic model (Soni and Nadi, 2019) on JavaScript and Python comments

	Category	P	R	F1	Supp.
JavaScript (Acc: 39.8%)	NO_URC	0.480	0.744	0.584	180
	URC_ADDRESSED	0.250	0.047	0.080	148
	URC_UNADDRESSED	0.129	0.184	0.151	49
Python (Acc: 40.1%)	NO_URC	0.513	0.669	0.581	148
	URC_ADDRESSED	0.412	0.070	0.120	100
	URC_UNADDRESSED	0.127	0.244	0.167	41

Thus, we conclude that, **unlike URCs, URC_UNADDRESSED are difficult to identify based on comment features.** In the future, more advanced models are needed to better capture the answer edit and comment post history.

6. Threats to Validity

In this section, we start with threats to external validity. Our empirical study is based on a dataset that contains 1,221 labeled comments extracted from a set of statistically sampled answer posts from Java questions. Although Java is one of the biggest communities in SO, and we can expect similar results for other popular languages, our findings may not be the same for questions on other topics.

Similarly, we only evaluate the performance of our models on sampled comments in accepted answer posts to questions relevant to Python, and JavaScript. Thus, the reported performance may not generalize to other communities. However, as the reported results across different categories are close in terms of accuracy, AUC and F1-score, we believe that our model should work for other communities.

Another threat to external validity is the limited size of empirical study and evaluation datasets due to the time-consuming annotation process. In the empirical study, we manually annotated a statistically significant sample of Stack Overflow answer posts and their comments. Specifically, We randomly sampled 384 answer posts, achieving a 5% margin error with a confidence level of 95%. To evaluate the proposed automated approaches, we created two test datasets by manually annotating 289 comments that were posted on 100 Python posts and 377 comments that were posted on 100 JavaScript posts. The performance of our models on the two datasets are close to each other. But still, the above results may not be applied to all Stack Overflow posts and comments.

As for threats to internal validity, the empirical study and the train/test of automated URC detection models rely on manually labeled comments and might be biased or error-prone. To avoid such a problem, two authors independently checked 100 comments and reported a high agreement ratio. Therefore we believe our coding guide is clear and could support future replication on other data.

7. Related Work

7.1. Studies on Stack Overflow Comments

The most relevant work to our study is by Soni and Nadi (2019) that analyzed how comments affect answer updates on Stack Overflow, although they did not perform any empirical study. They employed a heuristic rule-based

approach to classify comments into four categories. Since these categories are also determine whether a comment is asking for update, we clarify the relationship between their comment categories and our categories below to reduce ambiguities.

1. **WARRANT UPDATE**: A comment that warranted an update but an edit was not made to the answer. The equivalent of this class in our classification is URC_UNADDRESSED.
2. **UPDATE**: A comment that warranted an update and an edit were made to the answer. This class is equivalent to our URC_ADDRESSED.
3. **NO UPDATE**: A comment that did not warrant an update. It is equivalent to our NO_URC class.
4. **UNKNOWN**: All other comments that are just text, URLs, or discussion (e.g., “Thank you so much for this answer.”).

Soni and Nadi claimed that “*only a few (~4-5%) comments resulted in answer updates*” and “*More than a quarter (~26-29%) of the comments we studied across the five tags [java, python, javascript, android, php] require the answer to be updated, but are ignored by answer posters*”. From these statements, one can conclude that $\sim 84\text{-}88\%$ ⁹ of update request comments are unaddressed. Different from them, by manually analyzing 1,221 comments, we found that only 36.5% of URCs remain unaddressed after a year. Such inconsistent results may arise due to several reasons. First, their study scope is different from ours. They focused on the updates in the code block of an answer post and ignored the update in text. But in reality, answer owners sometimes only need to update the text to address URCs rather than touching code. We argue that text updates are also important because both text and code are important for a high-quality post (Calefato et al., 2015), and developers rely on both when utilizing SO (Wu et al., 2019; Chatterjee et al., 2020). Secondly, Soni and Nadi ignore the answers in the following comments. However, we found that 58.6% of our annotated URCs are addressed in the following comments. Thirdly, their findings rely on their rule-based approach to identify update-introduce comments, and their reported accuracy is 85% on labeled comments from 30 answer posts. Instead, we manually examined the studied dataset, which is more accurate.

⁹ $26/(26+5)=83.9\%$ and $29/(29+4)=87.9\%$

Though their categorization is different from ours, we can still compare with their heuristic approach in identifying URCs and unaddressed URCs. Our experiment results (ref. Table 7, 8, 9, and 10) show that our proposed machine learning-based approach outperforms their approach with a large margin. Moreover, while the algorithm by Soni and Nadi (2019) discards many comments due to either not including code updates or being labeled as UNKNOWN, our model labels all comments and doesn't discard any comment.

Another closed study on SO comments is by Zhang et al. (2021). They found that 4.4 million comments (possibly including informative comments) are hidden by default from developers. To help identify informative comments, they propose a taxonomy to group comments into seven types: Praise (praise an answer), Advantage (discuss the advantage of an answer), Improvement (make improvement to an answer), Weakness (point out the weakness of an answer), Inquiry (make inquiry based on an answer), Addition (provide additional information to an answer), and Irrelevant (discuss irrelevant topics to an answer). Their categorization is different from ours because our categorization focuses on whether a comment asks for an update on the answer post.

7.2. Other Studies on Stack Overflow

Studies on Stack Overflow can be categorized into two types, i.e., mining questions and answers on Stack Overflow to extract the challenges faced by developers (Treude et al., 2011; Barua et al., 2014; Rosen and Shihab, 2016; Ahmed and Bagherzadeh, 2018; Tahir et al., 2018; Bagherzadeh and Khatchadourian, 2019; Openja et al., 2020; Tan et al., 2020; Wen et al., 2021) and investigating the mechanisms used by Stack Overflow and proposing new feature/model to improve user experience on Stack Overflow (Xia et al., 2013; Saha et al., 2013; Nasehi et al., 2012; Asaduzzaman et al., 2013; Ponzanelli et al., 2014; Beyer and Pinzger, 2015; Zhang et al., 2015; Ahasanuzzaman et al., 2016; Srba and Bielikova, 2016; Yang et al., 2016; Mizobuchi and Takayama, 2017; Chen et al., 2018; Wang et al., 2018b,a; Zhang et al., 2019; Chatterjee et al., 2020).

Treude et al. (2011) manually categorized the types of questions on Stack Overflow, and observed that Stack Overflow could be useful for code review and learning the concepts of programming. Barua et al. (2014) first applied LDA, a popular statistical topic model, to discover topics from the contents

on Stack Overflow and track the changes of the topics over time. Following their methodology, researchers have analyzed contents on Stack Overflow related to fine-grained domains. For instance, Rosen and Shihab (2016) analyzed 13,232,821 posts to examine what mobile developers ask about. They discovered hot topics and determined what popular mobile-related issues are the most difficult. Ahmed and Bagherzadeh (2018) applied a similar methodology to analyze what do concurrency developers ask on Stack Overflow. Tahir et al. (2018) found that developers widely use Stack Overflow to ask for general assessments of code smells or anti-patterns instead of asking for particular refactoring solutions. More recent, Stack Overflow content related to big data analysis (Bagherzadeh and Khatchadourian, 2019), release engineering (Openja et al., 2020), bug severity (Tan et al., 2020), and serverless computing (Wen et al., 2021) are analyzed to help relevant stakeholders better understand the trends, challenges, and potential future development/research directions.

Many prior studies are investigating the quality of the crowd-sourced knowledge presented on Stack Overflow. Asaduzzaman et al. (2013) analyzed unanswered questions on Stack Overflow and found that the quality of questions is strongly related to whether a question receives an answer. Srba and Bielikova (2016) observed that an increasing amount of content with relatively lower quality is hurting the Stack Overflow community. Nasehi et al. (2012) examined code examples on Stack Overflow and identified characteristics of high-quality code examples. Yang et al. (2016) focused on the quality of code snippets on Stack Overflow. They examined the usability of code snippets by compiling or running them. Zhang et al. (2019) analyzed the obsoleteness of answers on Stack Overflow. They found that more than half of the obsolete answers were probably already obsolete when they were first posted. Moreover, when an obsolete answer is observed, only a small proportion (20.5%) of such answers are ever updated. Thus they suggest that Stack Overflow should develop mechanisms to encourage the whole community to maintain answers. Chatterjee et al. (2020) conducted an exploratory study of novice software engineers' focus in stack overflow posts. They found that Novice programmers focus on 15–21% text and 27% code in a Stack Overflow post.

Prior studies also examined Stack Overflow's mechanisms to understand its operation better and proposed tools to improve the efficiency of the knowledge-sharing process. For instance, to enhance the quality of knowledge on Stack Overflow, Ponzanelli et al. (2014) proposed an automated approach

to identify the quality of posts and filter low-quality content. Wang et al. (2018a) studied how Stack Overflow users revise answers and what is the impact of those revisions. They found that although the current badge system on Stack Overflow is designed to ensure the quantity of revisions, such a badge system fails to consider the quality of revisions and should be improved in the future. Chen et al. (2018) proposed a convolutional neural network (CNN) based approach to predict the need for post revisions to improve the overall quality of Stack Overflow posts. Several approaches are raised to automatically predict tags on Stack Overflow questions (Xia et al., 2013; Saha et al., 2013; Beyer and Pinzger, 2015; Wang et al., 2018b; Chen et al., 2019), and identify duplicate posts (Zhang et al., 2015; Ahasanuzzaman et al., 2016; Mizobuchi and Takayama, 2017).

8. Conclusion and Future Work

Comments on Stack Overflow answer posts act as a potential way to improve the quality of the answers, which is one main concern of Stack Overflow community. In this paper, we conduct a study on URCs (update request comments)—comments in answer posts that explicitly or implicitly ask for an answer update due to reasons such as warning issues in the answer. Specifically, we investigate what happens when a user posts a URC and how/when/by whom it gets addressed. For this purpose, we manually examine a sample set of 1,221 comments on answer posts of questions tagged with “java”. We find that 51.7% of the analyzed comments are URCs. Most addressed URCs (80.1%) are addressed by the answer owners, and more interestingly, most URCs (55.3%) are addressed within 24 hours. Nevertheless, 36.5% of URCs remain unaddressed after a year.

Upon checking the votes received by URCs, we find that the majority URCs have a score of 0, which may cause them to be invisible to community members who are potentially interested in addressing them. Thus, as the first step towards improving the awareness of URCs, we explore the feasibility of building a tool that can automatically identify URCs as they post. Such a tool can also be leveraged to mine URCs for research purposes. Specifically, we proposed a set of comment features for URC detection and trained several supervised models, including random forest and BERT from 1,221 annotated Java comments. We evaluated the performance of our models on Python and JavaScript comments. Experiments results show that our automated URC detector can identify URCs with around 90% accuracy.

In the future, we would like to increase the number of annotated comments for train and evaluation and investigate if specific kinds of URCs are more likely to be addressed. We also plan to analyze URCs that non-answer owner addresses to explore what types of SO users are more likely to help the community address URCs.

Acknowledgement

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number: RGPIN-2019-05071].

References

- Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K., Schneider, K.A., 2016. Mining duplicate questions of stack overflow, in: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), IEEE. pp. 402–412.
- Ahmed, S., Bagherzadeh, M., 2018. What do concurrency developers ask about? a large-scale study using stack overflow, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10.
- Asaduzzaman, M., Mashiyat, A.S., Roy, C.K., Schneider, K.A., 2013. Answering questions about unanswered questions of stack overflow, in: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE. pp. 97–100.
- Atwood, J., 2018. What does stack overflow want to be when it grows up. URL: <https://blog.codinghorror.com/what-does-stack-overflow-want-to-be-when-it-grows-up>.
- Bagherzadeh, M., Khatchadourian, R., 2019. Going big: a large-scale study on what big data developers ask, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), pp. 432–442.

- Baltes, S., Treude, C., Diehl, S., 2019. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets, in: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE. pp. 191–194.
- Barua, A., Thomas, S.W., Hassan, A.E., 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 619–654.
- Beyer, S., Pinzger, M., 2015. Synonym suggestion for tags on stack overflow, in: 2015 IEEE 23rd International Conference on Program Comprehension, IEEE. pp. 94–103.
- Breiman, L., 2001. Random forests. *Machine learning* 45, 5–32.
- Calefato, F., Lanubile, F., Marasciulo, M.C., Novielli, N., 2015. Mining successful answers in stack overflow, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 430–433. doi:10.1109/MSR.2015.56.
- Chatterjee, P., Kong, M., Pollock, L., 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software* 159, 110454. URL: <https://www.sciencedirect.com/science/article/pii/S0164121219302286>, doi:<https://doi.org/10.1016/j.jss.2019.110454>.
- Chen, C., Chen, X., Sun, J., Xing, Z., Li, G., 2018. Data-driven proactive policy assurance of post quality in community q&a sites. *Proceedings of the ACM on human-computer interaction* 2, 1–22.
- Chen, H., Coogle, J., Damevski, K., 2019. Modeling stack overflow tags and topics as a hierarchy of concepts. *Journal of Systems and Software* 156, 283–299. URL: <https://www.sciencedirect.com/science/article/pii/S0164121219301499>, doi:<https://doi.org/10.1016/j.jss.2019.07.033>.
- Gu, K., Budhkar, A., 2021. A package for learning on tabular and text data with transformers, in: Proceedings of the Third Workshop on Multimodal Artificial Intelligence, Association for Computational Linguistics, Mexico

- City, Mexico. pp. 69–73. URL: <https://www.aclweb.org/anthology/2021.maiworkshop-1.10>, doi:10.18653/v1/2021.maiworkshop-1.10.
- Hanley, J.A., McNeil, B.J., 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* 143, 29–36.
- Hosmer Jr, D.W., Lemeshow, S., Sturdivant, R.X., 2013. Applied logistic regression. volume 398. John Wiley & Sons.
- Loria, S., 2018. textblob documentation. Release 0.15 2, 269.
- May, A., Wachs, J., Hannák, A., 2019. Gender differences in participation and reward on stack overflow. *Empirical Software Engineering (EMSE)* 24, 1997–2019.
- Mizobuchi, Y., Takayama, K., 2017. Two improvements to detect duplicates in stack overflow, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE. pp. 563–564.
- Nasehi, S.M., Sillito, J., Maurer, F., Burns, C., 2012. What makes a good code example?: A study of programming q&a in stackoverflow, in: 2012 28th IEEE International Conference on Software Maintenance (ICSM), IEEE. pp. 25–34.
- Openja, M., Adams, B., Khomh, F., 2020. Analysis of modern release engineering topics:–a large-scale study using stackoverflow–, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 104–114.
- Ponzanelli, L., Mocci, A., Bacchelli, A., Lanza, M., Fullerton, D., 2014. Improving low quality stack overflow post detection, in: 2014 IEEE international conference on software maintenance and evolution, IEEE. pp. 541–544.
- Qu, C., Yang, L., Croft, W.B., Zhang, Y., Trippas, J.R., Qiu, M., 2019. User intent prediction in information-seeking conversations. *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval* .
- Rosen, C., Shihab, E., 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* 21, 1192–1223.

- Saha, A.K., Saha, R.K., Schneider, K.A., 2013. A discriminative model approach for suggesting tags automatically for stack overflow questions, in: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE. pp. 73–76.
- Soni, A., Nadi, S., 2019. Analyzing comment-induced updates on stack overflow, in: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE. pp. 220–224.
- Srba, I., Bielikova, M., 2016. Why is stack overflow failing? preserving sustainability in community question answering. *IEEE Software* 33, 80–89.
- Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., Counsell, S., 2018. Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, pp. 68–78.
- Tan, Y., Xu, S., Wang, Z., Zhang, T., Xu, Z., Luo, X., 2020. Bug severity prediction using question-and-answer pairs from stack overflow. *Journal of Systems and Software* 165, 110567. URL: <https://www.sciencedirect.com/science/article/pii/S0164121220300480>, doi:<https://doi.org/10.1016/j.jss.2020.110567>.
- Tang, H., Nadi, S., 2021. On using stack overflow comment-edit pairs to recommend code maintenance changes. *Empirical Softw. Engg.* 26. URL: <https://doi.org/10.1007/s10664-021-09954-8>, doi:[10.1007/s10664-021-09954-8](https://doi.org/10.1007/s10664-021-09954-8).
- Treude, C., Barzilay, O., Storey, M.A., 2011. How do programmers ask and answer questions on the web?(nier track), in: Proceedings of the 33rd international conference on software engineering, pp. 804–807.
- Wang, S., Chen, T.H., Hassan, A.E., 2018a. How do users revise answers on technical q&a websites? a case study on stack overflow. *IEEE Transactions on Software Engineering* 46, 1024–1038.
- Wang, S., Lo, D., Vasilescu, B., Serebrenik, A., 2018b. Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering* 23, 800–832.

- Wen, J., Chen, Z., Liu, Y., Lou, Y., Ma, Y., Huang, G., Jin, X., Liu, X., 2021. An empirical study on challenges of application development in serverless computing, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), pp. 416–428.
- Wu, Y., Wang, S., Bezemer, C.P., Inoue, K., 2019. How do developers utilize source code from stack overflow? *Empirical Software Engineering* 24, 637–673.
- Xia, X., Lo, D., Wang, X., Zhou, B., 2013. Tag recommendation in software information sites, in: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE. pp. 287–296.
- Yang, D., Hussain, A., Lopes, C.V., 2016. From query to usable code: an analysis of stack overflow code snippets, in: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), IEEE. pp. 391–401.
- Yang, Y., Xia, X., Lo, D., Bi, T., Grundy, J., Yang, X., 2020. Predictive models in software engineering: Challenges and opportunities. *arXiv preprint arXiv:2008.03656* .
- Zhang, H., Wang, S., Chen, T.H., Hassan, A.E., 2021. Are comments on stack overflow well organized for easy retrieval by developers? *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 1–31.
- Zhang, H., Wang, S., Chen, T.H.P., Zou, Y., Hassan, A.E., 2019. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering* .
- Zhang, W., Yang, Y., Wang, Q., 2011. Handling missing data in software effort prediction with naive bayes and em algorithm, in: Proceedings of the 7th International Conference on Predictive Models in Software Engineering, pp. 1–10.
- Zhang, Y., Lo, D., Xia, X., Sun, J.L., 2015. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology* 30, 981–997.