Data Analysis of Community Discussions on Security Issues

Gunardi Ali
gunardilin@mailbox.org
Harz University
Wernigerode, Germany

ACM Reference Format:
Gunardi Ali. 2023. Data Analysis of Community Discussions on Security Issues. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

Configured software is widely used to accommodate users with specific requirements. It promises user-friendliness and process automation, as it is developed based on specific user requirement. By implementing new requirements, the program developer needs to add or modify existing code. In the process, there is a likelihood of new vulnerabilities being introduced, even perhaps unknowingly by the developer. Implementing new functionality properly has its own challenge. Developers often rely on community question answering forum such as StackOverflow to get quick answers or consult with other programmers.

To date, there is still no empirical study analyzing security issues of configurable software using StackOverflow data. To address this gap we have applied a natural language processing method, i.e. Latent Dirichlet Allocation (LDA) to investigate a large amount of StackOverflow data from the last 10 years (2012-2021). The purpose is to find the recurring security topics of configurable software and to discover aspects of it that are under-researched. One of the key findings is **cloud access management** and **Android and Java development** are two dominant security topics in the context of configurable software and their trends are raising continuously in the last 10 years. Furthermore, there are indications, that security-related topics are under-researched. This study aims to understand the current trend of security-related topics and present undeservingly under-researched areas.

2 INTRODUCTION

Every organization in different industries creates, processes and accumulates data. In order to do it in an efficient manner, growing companies are investing in information processing software [10]. Thus they will be confronted with the question, whether they should use **off-the-shelf**, **customized** off-the-shelf, or **custom** software. Off-the-shelf software is generic built software that is commercially available for a wide range of consumers. It provides only standard or predefined frameworks that target a large number of users [8]. Modern off-the-shelf software often has application programming interfaces (APIs) that allow to some extent customization or extension on top of the base software. The scope of customization depends starkly on predefined frameworks [7, 12]. On the other hand, custom software is tailor-made to cater a specific requirement of an enterprise. Developing custom software is more expensive because it requires a team of experts consisting project manager, designer and software developer specifically to fulfill unique requirements [18]. Furthermore, after integrating the custom software, there will be an ongoing cost incurred because the developer is still required to fix bugs, add new functionality or improve existing frameworks [14]. From here on out, we refer customized and custom software as **configurable** software.

From developer's point of view, programming is an ongoing learning process to solve new problems. Consulting with other developers to help them solving problem might be ideal, but unfortunately it is not always practical. Hence they rely on a different knowledge base to find answers. One of the best ways is to look for an existing working example which could be a coding tutorial or code snippet for other similar problems [16]. A good source for code snippets is a community Q&A programming website like StackOverflow, Quora, Reddit and CodeProject.

Despite the importance of security-related topics, we are not aware of any qualitative study analyzing questions and answers posted on the relevant Q&A websites. To address this research gap we have applied text analysis using publicly available StackOverflow dataset[1] with a focus on security aspects of configurable software. There are two ways to retrieve information from text:

- Traditional method, which involves manually reading and investigating text documents. It is very time-consuming and therefore not a scalable approach for big amount of data.
- Natural Language Processing (NLP) method which programmatically allows a more scalable approach for topic discovery on a massive amount of text data. For topic discovery, the following techniques can be used: Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA) and Non-Negative Matrix Factorization(NNMF) [4].

In this paper, we applied data exploratory analysis and LDA algorithm[2] on 264.148 text data collected from StackOverflow. The data range is over 10 years (2012-2021). Based on this dataset we have addressed the following research questions (**RQ**):

RQ1: What is the general trend for security-related questions?
RQ2: What are the dominant topics and their trends in the last 10 years?

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://cloud.google.com/blog/topics/public-datasets/google-bigquery-public-datasets-now-include-stack-overflow-q-a?hl=en
[2] The source code used in this paper is available on: https://doi.org/10.5281/zenodo.7737030

RQ3: What is the reason that some questions do not have an accepted answer? What are their dominant topics?

Security is one of the most important pillars when creating a well-architected software system [1]. But that is not always the case which leads to many damaging consequences [2]. This aspect leads to the motivation for **RQ1** which is to investigate the developing trend of security in configurable software and get a general overview of the current state. To date, there is no qualitative study about security issues of configurable software using data from StackOverflow. Thus, discovering the dominant topics is the motivation for **RQ2**. Furthermore, there might be topics in this area that should get more attention. They could be identified by finding questions with recurring themes, which are still unanswered and getting few responses. Hence, the motivation for **RQ3** is to inform and raise the awareness of readers about topics that are getting less attention than they deserve. The findings might be useful for academic institutes as an orientation to redirect effort and necessary resources into under-researched, possible topics for future research.

3 BACKGROUND

In the following section, we provide background information about StackOverflow and security concerns of configurable software.

3.1 StackOverflow

Software developers confront coding problems on daily basis. Consulting with fellow developers is not always practical and can sometimes delay their progress. Hence, they rely more on multiple knowledge bases. E.g. if they are not familiar with an error message when using a specific API, they would open its official documentation or many other publicly available coding tutorials to find working examples. Another way to solve problems is to look for a working code snippet that solves other fellow developers' similar problems. This is only possible due to the existence of community Q&A programming websites.

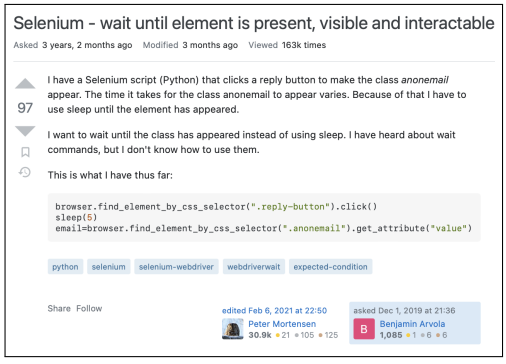

Figure 1: StackOverflow Question with Score 97.

StackOverflow[3] is the most popular Q&A programming website with 100+ million monthly visits on average[4]. It allows developers to post and answer coding questions. Before posting a question, they

[3] https://stackoverflow.com
[4] https://stackoverflow.co/advertising/audience/

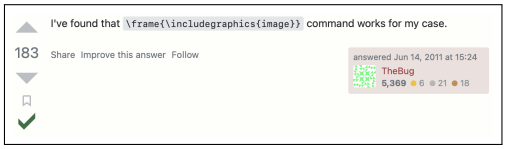

Figure 2: "Accepted Answer" with Score 183 and Green Tick.

are encouraged to search if the same question was posted before. If it exists and they still post it or if the problem is not described in an understandable manner, their question will be voted down by other developers. If other audiences find a question helpful, then the post will be up-voted. (See figure 1) In the same way, answers can also be up- or down-voted based on their correctness and helpfulness. Figure 1 shows a question with score 97. Score indicates the number of upvotes minus downvotes and will impact each user's reputation. Out of multiple answers, only one answer can be marked as an "accepted answer" with a green tick by the asking user, if it specifically solves the problem. (Refer to figure 2) Therefore a developer who seeks a code snippet solution for similar questions can directly check for questions and especially answers with the highest votes or the "accepted answer".

3.2 Security concerns of configurable software

When developing a configurable software either in-house or by using a third-party developer, an enterprise starts with specifying the requirements. Security is an important feature and must be stated explicitly as a requirement [19]. Unfortunately in most cases, this aspect is left out [2]. Followings are the reasons why it is difficult to develop a secure software:

(1) *Writing security features in requirements is like asking obvious questions* like if malicious malware should be prevented to exploit your web application. Hence they are often left out. Therefore it depends on the developer's goodwill to implement security features, which are not explicitly stated in the requirements [19].
(2) *Program development project could involve multiple vendors.* These different developers need to apply interfaces and protocols by using the same specifications [11]. Lacking well-defined specifications could lead to unwritten but necessary security features being ignored. Later on, especially when a problem comes up, it could lead to finger-pointing.
(3) *Software requirements mostly consist of functional and positive aspects*, which state what end users do want and not what they don't want. Therefore negative cases which include abuse, misuse and confuse cases are not taken into consideration during the development process. These problems are at best found during the post-development test [2, 19].
(4) *Fulfilling functionality requirement is the developer's first priority*, while the security aspect is completely ignored in extreme cases. Security vulnerabilities could be patched later as updates, hoping they are not exploited in the meantime. Some developers view this as a reasonable strategy [2].
(5) *Mismanagement*: Lack of security guideline in place and having to deal with competing priorities are the most frequent security deterrents [3].

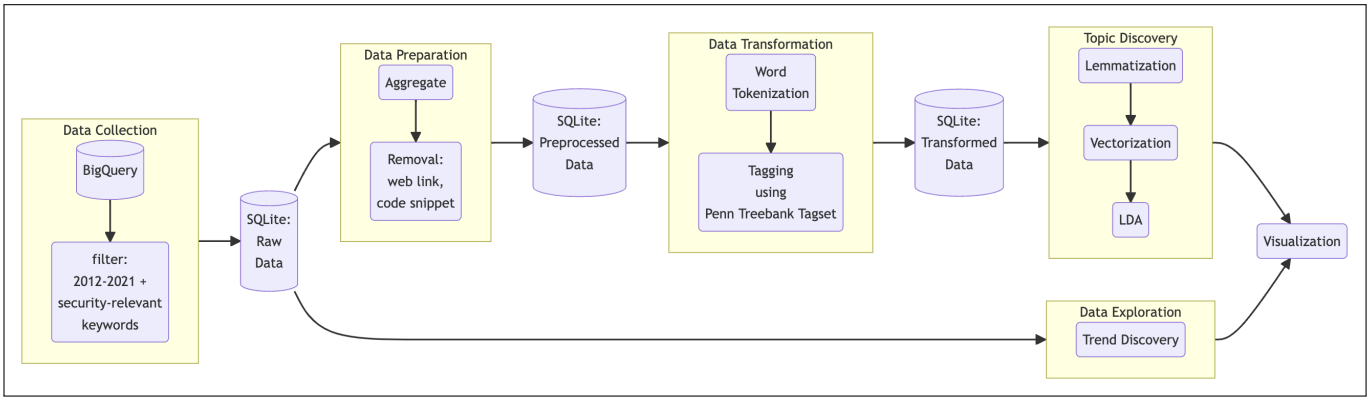

Figure 3: Methodological Overview of Our Text Analysis.

4 METHODOLOGY

Our objective in this paper is to find insight from StackOverflow dataset about security topics on configurable software. In following, we describe every executed step to achieve our objective. For a methodical overview, please refer to figure 3 above.

4.1 Data Collection

StackOverflow public dataset is available on Google BigQuery database. Due to the huge volumes of data available, we decided to use the following **filter** conditions to get relevant data for our text analytics:

(1) Last 10 years time frame. By the time of writing, the complete data for the year 2022 was not yet available, hence we used data from the years *2012-2021*.
(2) Search string for questions containing security-relevant topics on configurable software using the following keywords:

> *secur*, *protect*, *custom*, *config*, *variab*, *featur*, *plug*, *system family*, *product family* and *software family*.

After applying both filters, we obtained 264.148 text data. We stored this filtered raw data in a local SQLite database for practical reasons. E.g. During the development process, a script needs to be executed and improved multiple times and collecting data from the local database is faster than otherwise.

4.2 Data Preparation

The filtered raw data is in tabular form with multiple attributes, e.g. *id*, *title*, *body*, *tags*, *view count*, etc. *Body* is the string content of a StackOverflow question. As the attribute *title* and *body* are relevant to our analytics, we performed data **aggregation** to combine them together.

At this stage, the combined text for the title and body could include a link to a website and a code snippet. We performed link **removal** because they are not necessary for our analytics. Code snippets were also removed because a word used in syntax have a mostly different meaning than in normal human conversation. E.g. the syntax *for* in Python programming language signals the start of looping over a sequence. Now the texts are cleaned from unnecessary links and code snippets. We stored them in a local SQLite database, for easier data retrieval purposes.

4.3 Data Transformation

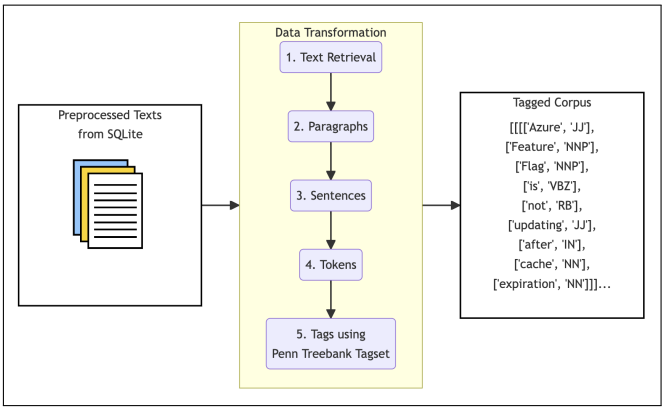

Figure 4: Breakdown of Transformation Process.

After the last stage, the text consists of a head and body. Furthermore, it still includes HTML tags (*p*, *h1*, *pre*, etc.). HTML tags consist information about the structure of the text data (paragraphs, sentences, words). We transformed our text data (corpus) by tokenizing it in a way that the result is compatible as input for the machine learning algorithm (in the next stage) while still preserving the text structure.

As can be seen in figure 4, we achieved this by applying the following steps to 264.148 preprocessed text:

(1) **Retrieve** each text one by one from local SQLite database. This step is necessary to ensure that the whole process is memory friendly.
(2) Break each text into **paragraphs**. This is possible by reading the HTML tags (*<p>*, *<h1>*, *<h2>*, *<li>*) which signal the start of a paragraph, title, subtitle or list.
(3) Break each paragraph into **sentences**. Capital letter and punctuation (. , *?*, *!*) signal the start or end of a sentence. We did not check these manually, because punctuation can

be used in different contexts and therefore it is not always used at the end of a sentence. E.g. period (.) can appear in floats or abbreviations [4]. Due to this complexity, we used a pre-trained model from the NLTK library (*sent_tokenize* method) to extract every sentence from a paragraph.

(4) Extract alphabetic and non-alphabetic characters/**tokens** from each sentence. A whitespace or non-alphabetic characters (punctuation) could mark the start of a new character. But this is not always straightforward as in the case with "*wouldn't*" [4]. To cope with this complexity, we applied another pre-trained model from NLTK library (*wordpunct_tokenize* method).
(5) Tag each character in the sentence based on its function in the sentence. The tag used is **Penn Treebank tagset**, which consists of 36 different tags. E.g. NN: singular noun, NNS: plural noun, VB: verb, JJ: adjective [15]. To achieve this, we applied a method from NLTK library, i.e. *pos_tag*.
(6) Repeat step 2-5 until all character in the text document retrieved in step 1 is processed. Afterward, all characters/tokens and its tag are stored in the local SQLite database.
(7) Repeat steps 1-6 until all 264.148 text are processed.

The input and output of the above steps can be seen in table 1 below. The transformed text in 3rd column still preserves the structure of paragraphs and sentences. It always starts with "*[[[[*", as the 1st bracket marks the start of a text document, 2nd a paragraph and 3rd a sentence. The 4th bracket contains a token and its Penn Treebank tag.

4.4 Topic Discovery

At this stage, all token is associated together with its Penn Treebank tag, e.g. ['headers', 'NNS'] or ['having', 'VBG']. For an exhaustive list of Penn Treebank tagset, refer to [15]. The associated Penn Treebank tag includes, among others, the following information:

(1) Token function in a sentence. E.g. noun: NN, verb: VB, adjective: JJ.
(2) If the token is a noun, the tag also includes information if it is in singular form *NN* or plural *NNS*.
(3) If the token is a verb, the tag differentiates between present tense *VB*, past *VBD* or gerund form *VBG*.

The tokens need to be normalized depending on their tag for dimension reduction purposes. This normalization needs to take the fact into consideration that one word can be used in different forms and contexts. In order to prevent information loss, similar words with different forms and contexts need to be normalized differently.

The examples in figure 5 show the complexity of normalization process:

(1) the words *gardening*, *garden* and *gardener* shouldn't be normalized to *garden*. It is better to normalize based on its Penn Treebank tag [4].
(2) the past form of *leave* which is *left* and the noun *left* (as in political *left*) shouldn't be normalized to *left*. The first one should become *leave*, while the second one stays as *left*.

To reduce this complexity, we applied **lemmatization** to normalize token based on its Penn Treebank tag. In other words, we

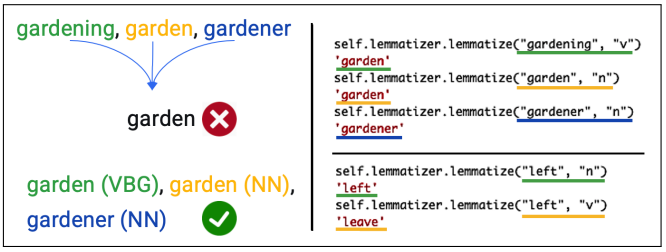

Figure 5: Different Normalization Approach (left); Normalizing Words by Lemmatization (right).

reduced the total number of word features. As shown in figure 5 above all token is normalized so that it still recognizes the subtle difference in similar tokens, e.g. the output still differentiates the noun *gardener* and verb *garden* [4, 5]. After lemmatization, the transformed text shown in the 3rd column of table 1 above becomes:

['spring', 'security', 'jwt', 'authenticate', 'via', 'header', 'send', 'jwt', 'trouble', 'try', 'get', 'spring', 'back', 'end', 'configure', 'correctly', 'try', 'send', 'jwt', 'react', 'app', 'user', 'logins', ...]

Box 1: Example of lemmatization output.

As can be seen in box 1, there are some words missing, e.g. *will*, *not*, *nor*, *I*, *have*, *been*. The missing words are removed because they are English stopwords. Furthermore, character + is filtered out from box 1, even though it is not a punctuation(„.?! , etc.). To remove such characters, we applied a rule to filter out strings with length: 1.

After lemmatization, the preprocessed data contains a list of texts, which represent each document. This preprocessed list is not compatible with machine learning (hereby ML) algorithm. Most ML method needs numerical representation as input. Therefore **vectorization** is needed to convert text to numerical representation for all documents. In general, there are three text vectorization methods [4]:

(1) *Frequency-based* counts how often a term is used in the text.
(2) *One-Hot Encoding* checks if a term appears in the text (1) or not (0).
(3) *Term Frequency-Inverse Document Frequency (TF-IDF)* starts by executing *frequency-based* method over all documents. Afterwards, it performs inverse calculation so that the most used term has the lowest weight and vice versa. The weights is normalized to be between 0 and 1.

We applied TF-IDF vectorization because by its nature it gives smaller weight to more frequent terms. If a term is used very often, it tends to offer little information about the context. In our study, the term *code* appears a lot and it makes sense for us to give such terms little weight.

In addition to that, we applied frequency filtering during vectorization to only allow terms that appear more than 20 times across all text documents. This is essential to reduce the complexity/dimension of our analytics. As can be seen in table 2, by applying a filter of minimum term frequency 20, there are 94% fewer terms

Table 1: Overview for Data Transformation.

Text ID	Before	After Transformation
71367783	<h1>Spring Security + JWT: Will not authenticate via headers nor send jwt</h1> <p>I have been having trouble trying to get my spring back end to configure correctly. ...	[[[['Spring', 'NNP'], ['Security', 'NNP'], ['+', 'NNP'], ['JWT', 'NNP'], [':', ':'], ['Will', 'MD'], ['not', 'RB'], ['authenticate', 'VB'], ['via', 'IN'], ['headers', 'NNS'], ['nor', 'CC'], ...

Table 2: Frequency Filter Comparison between 0 and 20 on Dataset for Year 2021, i.e. 24.423 text documents.

Frequency Filter	Total Term
0	66.454
20	4.071

than without a frequency filter. Fewer features mean faster computing performance for follow-up ML processes.

The result of vectorization, which is a numerical representation for each text document can be used as input for an ML method. We applied an unsupervised ML method, i.e. **Lattent Dirichlet Allocation (LDA)** to discover topics in our text documents. LDA is a generative probabilistic model, in which the topic is represented as the probability that each of a given set of terms will occur [6]. The text document is represented as a probability mixture of these topics [4, 6]. In other words, the topic of document (*d*) does not require to be distinct (100% topic *A*) but a mixture of different topics (P(Topic_x|Document_d): 34% for topic *A*, 56% *B*, 10% *C*, etc.) and a term (*tree*) can appear in multiple topics (P(Topic_x|Term_y): 56% for topic *A*, 34% *D*, 10% *E*). For more clarity, refer to figure 6.

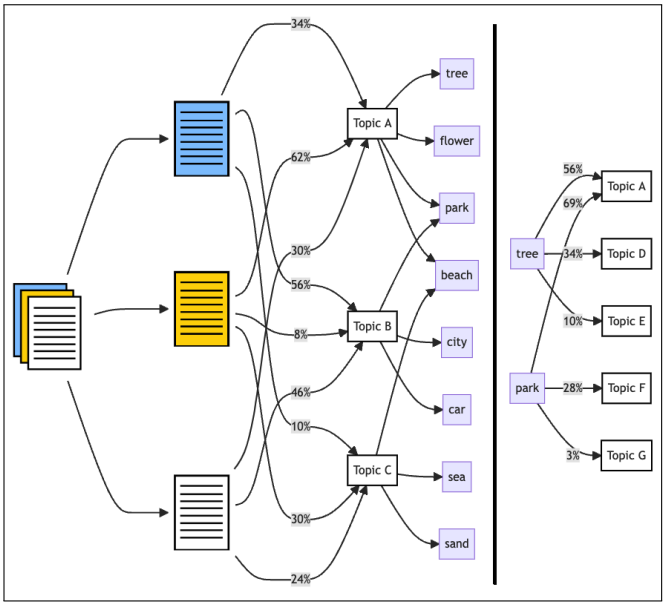

Figure 6: Simplified Overview of LDA: a document can be represented as a probability mixture of different topics (left), a topic consists of multiple words and a word can appear in multiple topics with different weights (right).

In order to generate a good LDA model, it is important to find the optimal value for the following hyper-parameters:

(1) Total number of topics (*n_topics*). It is done by checking coherence values for different *n_topics*. Basically, in this step, we generated LDA model for each *n_topics* (in our case from 3 until 20) and compared the coherence value for all generated LDA models. Each of these LDA models is generated based on our 10 years of data. The optimal *n_topics* would be the one with the highest coherence value [17]. Based on figure 7, the n_topics with the highest coherence score is 7.

(2) The total number of iterations (in other words: How often can a model read the data again?). Ideally, the model can read the data as often as possible until the model converges on every available document and afterward stop reading. In library *Gensim*, it could be controlled by modifying the following hyper-parameters: *passes* and *iterations*. [5]

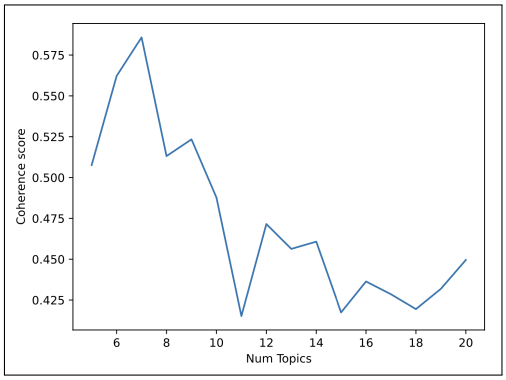

Figure 7: Coherence Value for Different Total Number of Topics (n_topics). Higher is Better.

These steps are computationally expensive and time-consuming, because it involves a lot of repeating model generation based on 10 years of data with a different set of hyper-parameters. Therefore we made sure to store all the models and supporting files after every repetition.

4.5 Visualization

After generating the optimal LDA model with *n_topics* 7, we visualized our model's result. For interactive topic model visualization, we used a Python library called *pyLDAvis*. As can be seen in figure 8, 5 out of 7 topics are overlapping each other. This result is not favorable to our analysis, because these topics are not unique enough from each other. Therefore we decided to inspect the LDA model

[5] https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html

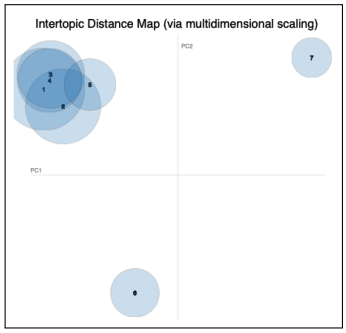

Figure 8: Unfavorable Distribution for 7 Topics.

with the second-highest coherence score. As can be seen in figure 7, the second best n_topics is 6 and the result can be seen in figure 9. It shows good topic distributions with few overlaps. In the right half of the figure, we can see the terms which belong to topic 3. With these terms, we interpreted each of the topics and presented our findings in section 5.2.

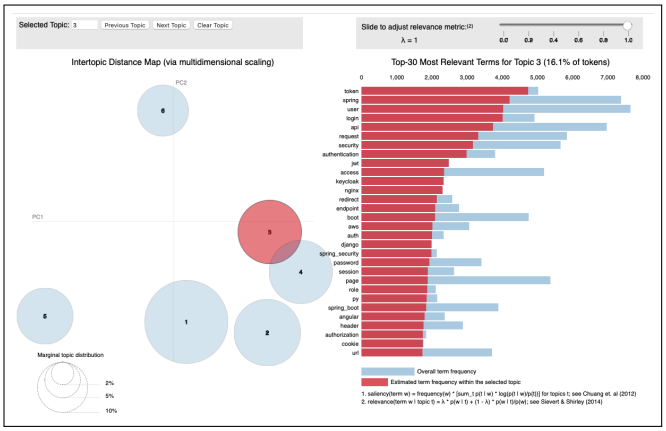

Figure 9: Visualisation for n_topics=6 using pyLDAvis.

5 RESULTS

In this section, we describe our findings and address our research questions. We would like to remind the reader that our findings are only specific to security-related questions and don't necessarily reflect the general overall trend of StackOverflow.

5.1 RQ1: What is the general trend for security-related questions on StackOverflow?

As can be seen in figure 10, the annual total questions was increasing from 17.834 and reached a peak in 2015 at 31.368. Afterwards, the total number was decreasing and stabilized at 24.501. The year with the highest number of questions without an accepted answer is 2016 with a total of 18.071. Our expectation was the most current year (2021) should have the most questions without accepted answer, because an older question would have a higher probability of getting

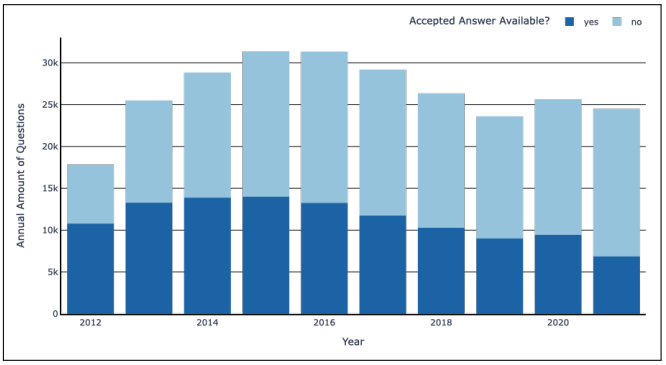

Figure 10: Total Questions: With vs. Without Accepted Answer.

an accepted answer, since more time has passed. But the fact that 2016 has the highest number without accepted answers can be explained by the very high total number of questions for 2015 and 2016.

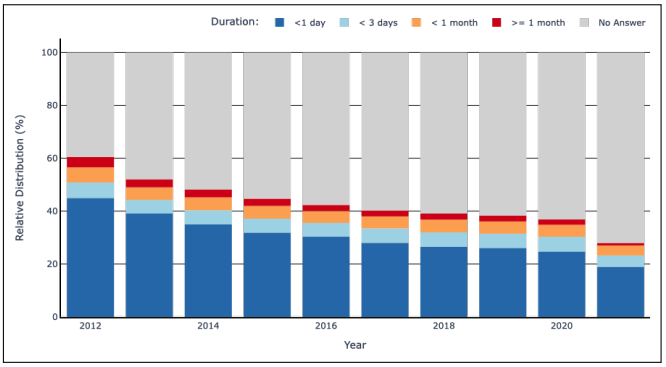

Figure 11: Duration Distribution for Security-Related Questions.

Figure 11 shows the annual distribution for the duration a question getting an accepted answer. It highlights the probability a question gets an accepted answer under the same time span (e.g. <1 day, etc.) is decreasing over the year. To get more validation about this trend, we chose the years 2013 and 2020 for our next analysis, because they have almost the same total questions.

Table 3: Frequency of a Security Related Question Getting an Accepted Answer Within a Month: 2013 vs. 2020.

Year	Total Questions	Questions with Accepted Answer (<1 Month)
2013	25.500	49%
2020	25.624	35%

Table 3 highlights our previous finding that the probability of a question getting an accepted answer under the same time span (e.g. <1 month) is decreasing over the year. Several factors could

contribute to this trend, e.g. the topics could be under-researched or few user engagement, which would be addressed in RQ3 section 5.3.

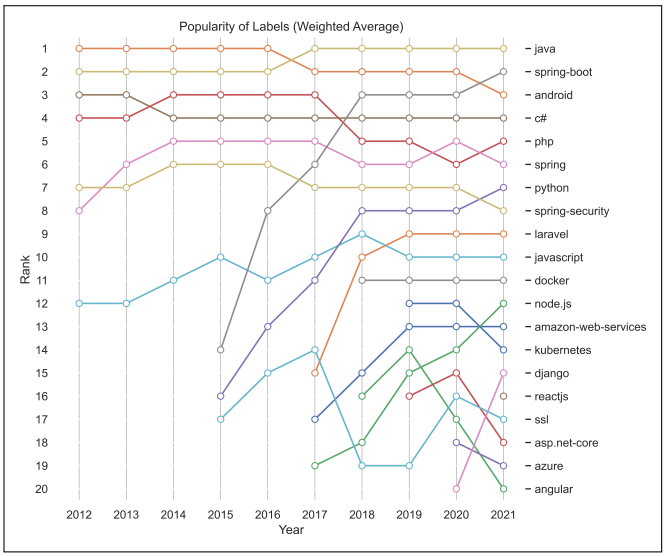

Figure 12: Label Popularity Rank (weighted average, as most questions contain multiple labels).
For simplicity, only top 20 labels of 2021 are shown in all ranking tables.

Figure 12 shows the top 20 labels used from 2012 - 2021. Label *java*, *android*, *c#* and *php* are in the top 5 during this time span. Spring-related labels (*spring* and *spring-security*) were in the top 10 over the year, while the label *spring-boot* was gaining traction, especially since the year 2014 from rank 39th to 2nd in the year 2021. Following labels were also becoming more popular: *python*, *laravel*, *docker*, *amazon-web-services*, *azure*, *node.js*, *reactjs*, *django* and *kubernetes*.

5.2 RQ2: What are the dominant topics and how are their trends in the last 10 years?

In this section, we address our findings from the topic discovery LDA algorithm. Instead of using the *n_topics* 7 which has the highest coherence value (refer to figure 7), we use *n_topics* 6. This is due to less overlapping clustering with *n_topics* 6. For more clarity, compare the overlapping in figure 8 and 9.

The result of the LDA model can be seen in table 4 under the 3rd column. The column shows different word combinations for 6 topics. Based on the combination, we interpreted the topics, which can be seen under the 2nd column.

We show the annual distribution for each topic in figure 13. As can be seen, the 2nd and 3rd topics (i.e. *Android & Java Development* and *Cloud Access Management*) are becoming more frequent. We analyzed the label's popularity specifically for questions without accepted answers. We didn't show the rank table, because the difference with the rank table from figure 12 is very minimal.

For the next section, we put our focus on 2nd and 3rd topic. As can be seen in figure 14, these topics have the highest absolute number

Table 4: Topic Interpretation Based on Word Combination (From LDA Output).

Index	Topic *(Interpretation)*	Words
1	General System Development	class, variable, value, data, function, model, object, method, table, like
2	Android & Java Development	docker, run, java, xml, spring, jdk, container, version, javax, kafka
3	Cloud Access Management	token, spring, user, login, api, request, aws, authentication, jwt, access
4	Server Architecture	azure, certificate, server, service, net, connection, client, ssl, connect, http
5	Mobile App Development	android, activity, app, button, firebase, flutter, java, android_studio, xamarin, mainactivity
6	Website Development	php, laravel, email, html, website, file, react, form, upload, wordpress

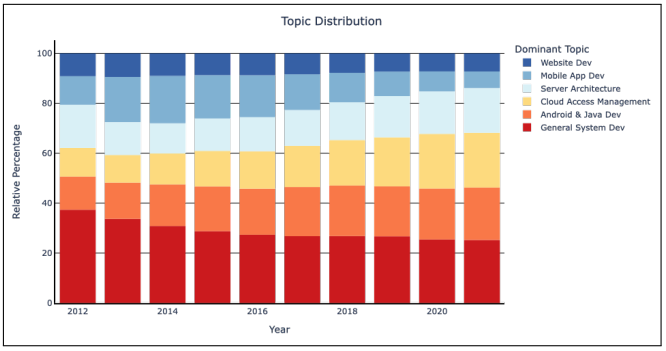

Figure 13: Topic Distribution from LDA Model.

without accepted answers. We ignored 1st topic because this topic represents a general programming question with no specific theme.

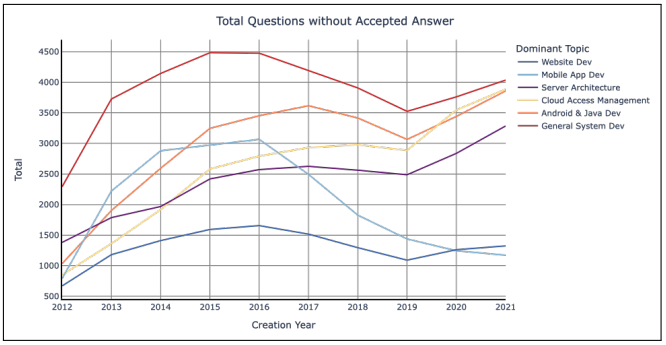

Figure 14: Total Number of Questions without Accepted Answer.

5.3 RQ3: What is the reason that some questions do not have an accepted answer? What are their dominant topics?

In this section, we start by presenting the dominant labels for 2nd and 3rd topics. Afterward, we compare the trend for security-related questions with other topics from StackOverview. The goal is to find if the security trend corresponds to the general trend in StackOverview, i.e. are security-related topics under-researched or does it reflect the general trend that questions get fewer answer? (due to less user engagement)

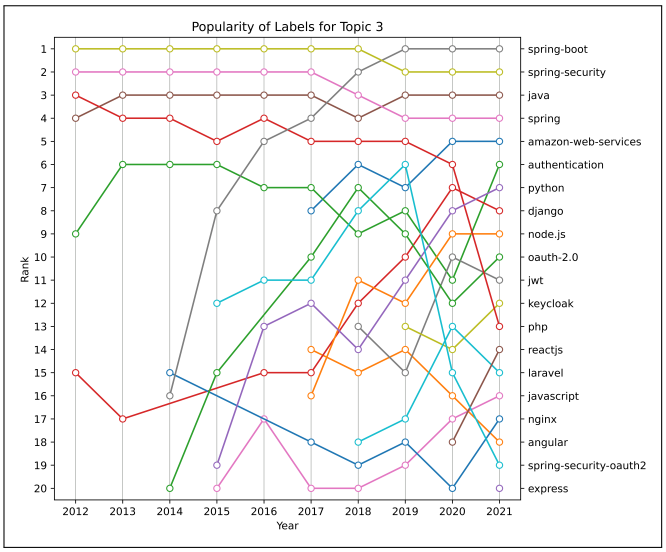

Figure 15: Label Popularity for Questions without Accepted Answer (Topic 3: Cloud Access Management).

Figure 15 reveals that spring-related themes are dominant in the 3rd topic, due to the following labels: *spring-boot*, *spring-security*, *spring* and *spring-security-oauth2*. Furthermore, the theme *Access Management on the Cloud* is dominant, due to the following labels: *amazon-web-services*, *authentication*, *oauth-2.0*, *jwt*, *keycloak* and *spring-security-oauth2*.

In 2nd topic, label *spring-boot* and *spring* are dominant but not as strong as in 3rd topic. Topics regarding *Container and its Deployment Tool* are also dominant, due to the following labels: *docker*, *kubernetes* and *docker-compose*. Furthermore, it also includes a topic about *Automation Tool for Deploying and Developing Program* with the following labels: *maven*, *apache-kafka* and *jenkins*.

Table 5: Frequency of All Question (Including Non Security-Related Topics) Getting an Accepted Answer Within a Month: 2012 vs. 2021.

Year	Total Questions	Questions with Accepted Answer (<1 Month)
2012	1.629.386	65%
2021	1.629.580	40%

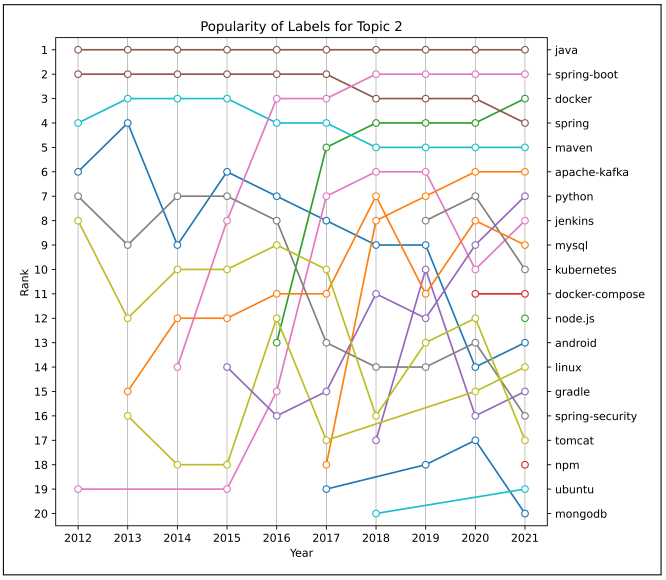

Figure 16: Label Popularity for Questions without Accepted Answer (Topic 2: Android & Java Development).

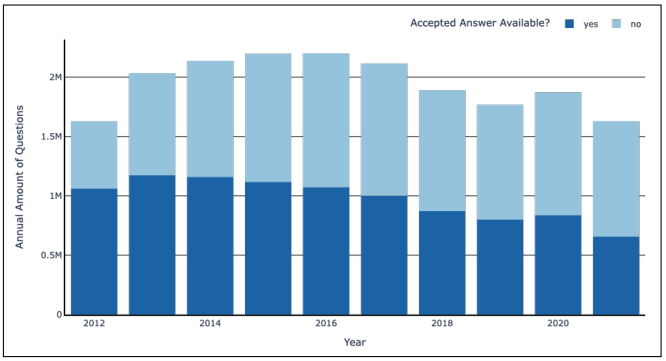

Figure 17: Accepted Answer Distribution for All StackOverflow Questions (including non security-related topics).

In figure 17 and 18 we show you the general trend for all questions (including non-security-related) on StackOverflow. The total number of questions was peaking in the year 2016 and went down to around the level from year 2012. Figure 18 reveals the relative distribution of questions getting an accepted answer was decreasing over the year. Table 5 highlights our previous finding that the probability of a question getting answered within 1 month is falling over 10 years. However, this downward trend was stronger especially for security-related questions, as can be seen in figure 19.

Figure 20 reveals that starting from the year 2014 the view count/question for security-related topics are on average higher than all topics on StackOverflow. Furthermore, since 2014 more security-related questions are bookmarked as *favorite* than all topics, while the average score for security questions has since been higher. (Refer to Figure 21) Score indicates the number of upvotes

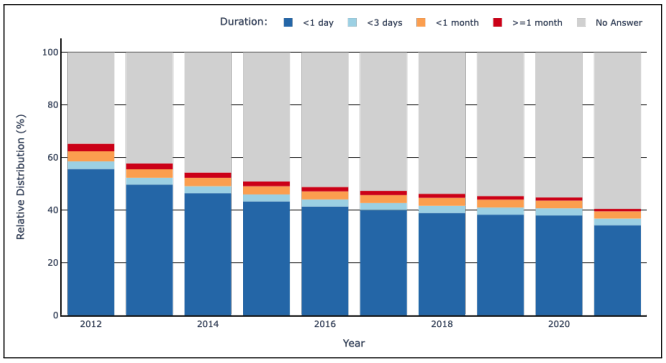

Figure 18: Duration Distribution for All StackOverflow Questions (including non security-related topics).

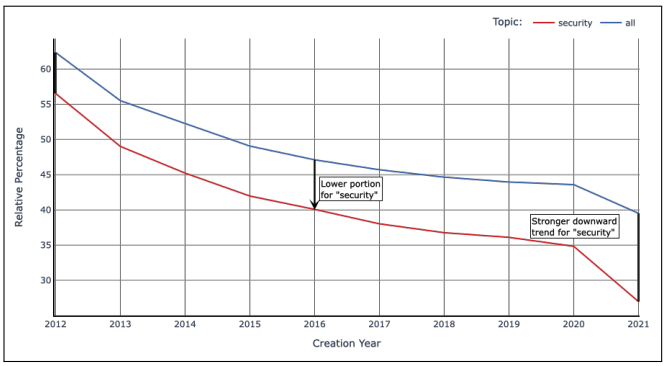

Figure 19: Probability of Getting Accepted Answer Within a Month: Security-Related vs. All StackOverflow Questions.

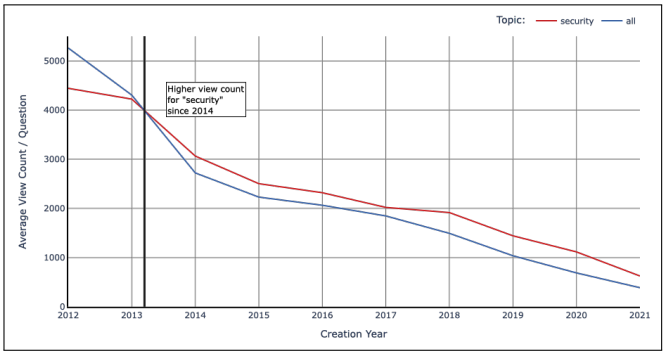

Figure 20: Average View Count: Security-Related vs. All StackOverflow Questions.

minus downvotes. In other words, the interest in security-related topic is higher than average for all topics.

The average comment number which can be seen in figure 22 supports this finding, as the average comment for security-related topics is higher than for all topics.

The average answer number for security-related topics is lower than for all topics. It could indicate that even with higher interest (i.e. view and comment count) for security-related topics, fewer

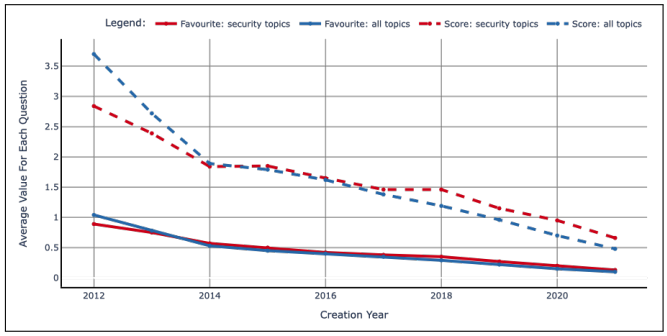

Figure 21: Average Favorite Count and Score: Security-Related vs. All StackOverflow Questions.

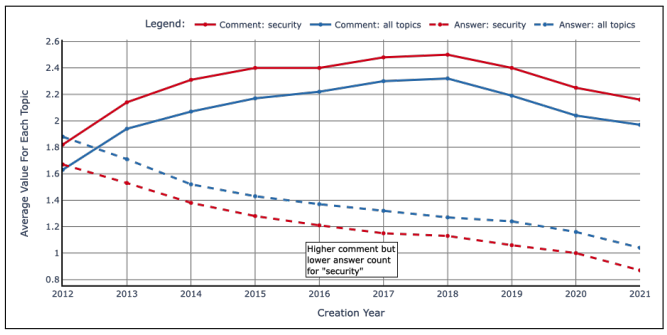

Figure 22: Average View and Comment Count: Security-Related vs. All StackOverflow Questions.

developers can deliver answers. Therefore, a smaller portion of security-related questions was getting accepted answers which can be seen in figure 19.

Based on these findings, we conclude that the interest in security-related topics is higher than average for other topics on StackOverflow. But it seems to be under-researched, as fewer developers can deliver accepted questions or even a question.

5.4 Analysis

Based on our previous findings (**RQ1**, **RQ2** and **RQ3**), we have concluded the following:

- There are indications that the questions were getting more difficult. Therefore, it took longer to get an accepted answer even though the user engagement (average total comment) was increasing over time.
- *Java* programming language, *Spring* framework (built on Java) and its extension *Spring Security* are very popular, while the popularity of *Spring Boot* is growing at a very fast pace.
- The popularity of *.NET* and its extension *ASP.NET* sank steadily, while *ASP.NET Model View Controller* fell rapidly over the time. It doesn't have to mean that they fell out of favour. It could indicate that they became mature and had therefore fewer vulnerability.

6 DISCUSSION

In this section, we presented some factors which could impact the validity of this paper.

6.1 Threats to Validity

We identified some threats that could impair the validity of our analysis. **First**, we could not analyze 841 out of 264.148 questions which are around 0.3% of all the data. This is due to some word combinations in the questions being unable to be processed during the step *Data Transformation*. (The process overview can be seen in figure 3.) Therefore we excluded these from our analysis. **Second**, in order to get security-related data, we applied keyword filtering, as described in section 4.1. One of the used keywords is *secur*. We found a question example, where the word *secur* appears but not in the context of *software security*. For a preview of that question, refer to figure 23. The word *secur* appears as a dataset about *food security* being used.

Figure 23: Question Containing Word *secur* Not In Context Of *software security*.

Third, it could be that our findings based on data from StackOverflow don't reflect the general trend for security-related topics. To overcome this, data from other community Q&A forums (such as Quora, Reddit, CodeProject, etc.) should also be included, to get a better picture of the overall security trend. **Fourth**, we set the limit for our paper to only analyze data from the last available 10 years (2012-2021). Hence, our findings might not reflect the trend for earlier years.

7 RELATED WORK

While reviewing for relevant literature, we found some studies analyzing security related questions on StackOverflow. None of the previous studies examined the general security-related trends and recurrent topics in context of configurable software. Lopez and Tun analyzed the security-related conversations between question asker and commenter. They found indications that StackOverflow interactions are shaping perceptions about security for non-specialist developers [13]. Another study from Yang and Lo performed a topic discovery algorithm using LDA on 30.054 StackOverflow posts (89% less than our study). However, the focus of the study is not on configurable software but rather general security topics [20]. Croft et al.'s study about security challenge of different programming language examined developers' discussions from Stack Overflow and GitHub. They found that the security-related discussion trend on StackOverflow and GitHub are different than each other. E.g. Web development security and authentication challenges are more popular on GitHub [9].

8 CONCLUSSION

In this paper, we presented our findings about recurring security-related topics in context of configurable software and their general trend. We did that by collecting security-related questions from StackOverflow and performing an unsupervised topic discovery algorithm, i.e. Latent Dirichlet Allocation. We found that the interest in topics *Android & Java Development* and *Cloud Access Management* are increasing during our analytic time frame (2012-2021). Furthermore, the interest or user engagement for security-related topics are higher than for general software development topic. However, on average security-related questions get fewer answers, which could indicate that this topic is in general under-researched.

REFERENCES

[1] Amazon Web Services, Inc. AWS well-architected framework. https://docs.aws.amazon.com/pdfs/wellarchitected/latest/framework/wellarchitected-framework.pdf#welcome, 2022.
[2] Hala Assal and Sonia Chiasson. Security in the software development lifecycle. In *SOUPS@ USENIX Security Symposium*, pages 281–296, 2018.
[3] Hala Assal and Sonia Chiasson. 'think secure from the beginning': A survey with software developers. Association for Computing Machinery, 2019.
[4] Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning*. O'Reilly Media, Inc., 1st edition, 2018.
[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
[6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
[7] Daniela Borissova and Ivan Mustakerov. A framework for designing of optimization software tools by commercial api implementation. *Int. Journal of Advanced Engineering, Management and Scienc*, 2:1790–1795, 10 2016.
[8] Evan Christiawan. Comparative assessment of custom developed and packaged software in accomplishing business's objective, March 2016.
[9] Roland Croft, Yongzheng Xie, Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. An empirical study of developers' discussions about security challenges of different programming languages. 07 2021.
[10] Feiqi Huang and Miklos A Vasarhelyi. Applying robotic process automation (rpa) in auditing: A framework. *International Journal of Accounting Information Systems*, 35:100433, 2019.
[11] Ronald Jabangwe, Darja Šmite, and Emil Hessbo. Distributed software development in an offshore outsourcing project: A case study of source code evolution and quality. *Information and Software Technology*, 72:125–136, 2016.
[12] Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. A systematic review of api evolution literature. *ACM Computing Surveys (CSUR)*, 54(8):1–36, 2021.
[13] Tamara Lopez, Thein Tun, Arosha Bandara, Levine Mark, Bashar Nuseibeh, and Helen Sharp. An anatomy of security conversations in stack overflow. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 31–40. IEEE, 2019.
[14] Ruchika Malhotra and Anuradha Chug. Software maintainability: Systematic literature review and current trends. *International Journal of Software Engineering and Knowledge Engineering*, 26(08):1221–1253, 2016.
[15] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.
[16] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012.
[17] Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408, 2015.
[18] John H Sinard and Peter Gershkovich. Custom software development for use in a clinical laboratory. *Journal of pathology informatics*, 3(1):44, 2012.
[19] Edward van Deursen and Jan Jaap Cannegieter. Building in security instead of testing it in. *Requirements Engineering Magazine*, April 2015.
[20] Xin-Li Yang, David Lo, Xin Xia, Zhiyuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31:910–924, 09 2016.