Mining Software Testing Knowledge from Stack Overflow

Dibyendu Gupta

Supervisor(s): Andy Zaidman, Baris Ardic

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Dibyendu Gupta
Final project course: CSE3000 Research Project
Thesis committee: Andy Zaidman, Baris Ardic, Koen Langendoen

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

This paper aims to unveil and gather testing-related information from Stack Overflow, highlighting it as a valuable resource for practitioners seeking answers and guidance. The study aims to accumulate knowledge from real-life experiences shared on Stack Overflow and bridge the knowledge gap between industry practices and teaching practices. The paper explores different types of software testing, popular frameworks, temporal trends of testing-related technologies, controversial opinions, and recommended practices/advice/suggestions from Stack Overflow posts. The methodology involves determining search terms through literature, querying the Stack Exchange API, conducting frequency analysis of words from posts, and manually inspecting threads. Our results show that the most popular frameworks discussed are Selenium, Spring, JMeter, and React. Automated testing and JavaScript frameworks have shown an upward trajectory over the years. The recommendations made by practitioners were categorized based on the broad scope of topics covered. We draw comparisons and parallels with related previous research and discuss the technical limitations faced during the study. Overall, this paper uncovers valuable insights from Stack Overflow and provides practitioners with the current view on industry practices.

1 Introduction

Software testing is a critical process in software development. Identifying defects and bugs in the early stages of software development prevents glitches and failure, detects security vulnerabilities, improves the quality of products, increases customer satisfaction and trust, helps with scalability and saves money in various situations [1]. Acknowledging the importance of software testing, research states that educating practitioners on software testing has often been inadequate [2]. Practitioners look for resources on the internet to answer their testing-related questions [3]. A widely recognized forum within the computer science community is Stack Overflow (SO[1]). SO is a question-answer forum for practitioners created in 2008 by Jeff Atwood and Joel Spolsky [4], designed for discussions and sharing knowledge between peers.

According to the 2022 survey conducted by Stack Overflow, 70% of the respondents have completed some form of higher education (a Bachelor's degree being the most common) and 80% of respondents are professionals [5]. This shows that SO is a good resource predominantly used by practitioners looking to clear their doubts on topics of interest and learn about new technologies. The survey also mentions that SO is the 2nd most relied-on online resource for learning how to code.

The objective of this study is to accumulate knowledge from the real-life experiences of practitioners. This knowledge should help educators identify the software testing topics that are unclear, unaware or misunderstood by practitioners. The objective is to bridge the knowledge gap between industrial practices and teaching practices by providing insights into the topics discussed on SO. From the stated research direction, the following research questions were formulated:

RQ1: ***What kind of information regarding types of software testing is available on stack overflow?***

1. ***What are the different types of software testing?***
2. ***What are the popular testing frameworks discussed on SO for different types of software testing?***
3. ***What are the temporal trends concerning popular technologies/frameworks/tools on SO?***
4. ***What are controversial software testing opinions on SO?***

There is a plethora of information that can be gathered from SO. Practitioners use SO to answer and discuss the practical problems they face with various technologies and tools. Answering what technologies and tools are used and how much their importance has changed over time is critical information to look at. This research question aims to create a better understanding for researchers and professionals about which topics are of relevance and could be included in higher education.

RQ2: ***What are the recommended practices, advice and suggestions on software testing from stack overflow?***
SO users often recommend good practices/advice on how other practitioners can improve their skills or what the standard practices are in industries. Numerous well-informed insights could be found in the domain of software testing. Aggregating these recommended practices will allow us to give an overview of the relevant practices and nuances related to software testing.

The paper is structured in the following sections: Section 2 discusses related work. Section 3 presents the methodology used for this research. Section 4 provides the results obtained and meaningful insights. Section 5 discusses some interesting insights, limitations and threats to validity. Section 6 contains the conclusion of the research paper and the potential of future research. Finally, Section 7 indicates how the research was performed responsibly.

2 Related Work

Looking at previous research enables us to adopt techniques used to gather and analyze valuable data from SO. Other research papers on software testing were explored to improve our knowledge of the topic. Previously published research has mined information from different domains of software testing and curated and analyzed questions from SO.

Kochhar has researched the topic of mining testing knowledge from Stack Overflow between 2009 - 2014. The research yielded knowledge about important topics of discussion, temporal trends, and technical challenges faced by developers. The results from this research state that testing frameworks, databases, threads, forms, builds, etc. are among

[1] SO is the acronym used for Stack Overflow through the paper.

the commonly discussed issues on SO. Among these categories, testing frameworks, databases and client servers were the "hot topics" at the time of the research. Furthermore, mobile development affiliated questions have seen an uptrend in recent years [6].

Openja et al. researched the release engineering (releng) sub-topic of software engineering from SO. Their research revolves around the types, popularity, and answerability of topics related to release engineering. The main findings show that most questions revolve around continuous integration/continuous deployment (CI/CD). They also discover the negative correlation between the popularity and difficulty of topics, meaning that difficult topics are less likely to be popular [7].

Bretschneider et al. has conducted research about the methodological requirements that identify "best practices". Their research provides a framework on how to determine "best practices" and reviews several statistical approaches that empirically identify "best practices" [8].

Allamanis and Sutton conducted research using topic modelling to associate programming concepts and identifiers to types of questions from Stack Overflow. Their research brings forth important conclusions about how topic models provide intuitions about the programming languages and problems faced by practitioners and how questions are similar across various programming languages [9].

Ardic and Zaidman have researched software testing education by examining higher-education curricula and surveying industry experts. Their research identifies the presence of dedicated software testing courses in universities, what topics practitioners demand, and how these practitioners acquire their testing knowledge and skills. Their research highlights the need to improve courses and practices of software testing for practitioners [10].

Garousi et al. explores the disjoint nature of collaboration between industry and academia focused on software testing. Their work is based on practitioners' opinions about the challenges faced while working and topics they want the research community to investigate [2]. This research provides the human perspective on the knowledge gap stated for this research..

3 Methodology

As of May 2023, there are over 24 million posts on SO. Hundreds of questions and threads are added to SO daily [11]. The continuous improvement of software and frameworks along with the influx of information and opinions on SO makes it critical to research on the latest available data. Recent data enables the research to be relevant amidst constant change and portrays an up-to-date image of the software testing community. This is done by discussing currently used frameworks and temporal trends in the software testing realm.

Openja et al. procured the dataset used in their research from SOTorrent [12]. They used the data dump from 2019 to circumvent the gathering of data. However, the latest data dump available on SOTorrent is from 2019. Working with a 4-year-old dataset contradicts the objective of this research of delivering the latest information available. With the reduced relevance of older data, it was decided to query testing-related data from January 2017 to May 2023. To gather definitive data and conduct reliable research, it is essential to devise and follow a proper methodology. The following section entails the process of choosing the search terms, creating datasets, pre-processing the data and analyzing it.

3.1 Listing the search terms

SO is a search-based website. Gathering data begins with identifying what search terms to query. This was done by gathering search terms from the literature and cross-referencing these terms from available online resources. According to ISTQB, types of software testing are different testing activities performed at different levels of the software development life cycle (SDLC) [13]. However, it is difficult to gather consensus about a definitive categorization of the different types of software testing (even for the ISTQB). Previous research is used to determine the different types of software testing used to create a list of search terms to answer RQ1.1. Ardic and Zaidman and Silva et al. conducted research that partly discussed the different types of software testing [10][14]. These terms along with the IEEE categorization of types of software testing enabled us to compile an extensive and relevant list of search terms [15]. These search terms were verified from online resources [16][17][18]. The search terms are stated below.

Types of software testing search terms:
Acceptance Testing, Compatibility Testing, Database Testing, DB Testing, End To End Testing, e2e Testing, Endurance Testing, GUI/UI Testing, Integration Testing, Load Testing, Performance Testing, Regression Testing, Security Testing, Spike Testing, Stress Testing, System Testing, Unit Testing

The concept of "best practices" is subjective. Bretschneider et al. [8] conducted research that examined the underlying assumptions behind best practices and was able to establish a set of rules to frame research designs for best practice studies. The rules revolve around completeness, randomness and comparability of the dataset. During this research, we were mindful to align our research with these rules. We also utilized Beyer and Pinzger's [19] research to understand how SO tags assign synonymous words under the same category. Their research provided a renewed perspective on how to formulate synonyms of best practices that would enable us to gather maximum coverage of posts and hence completeness of the dataset. We use this research to formulate our search terms for RQ2.

Search terms for RQ2:
Advice/Advise, Best approach, Benchmark, Best practices, Good practices, Smart practices, Standard, Software testing, Suggestions, Tips

3.2 Gathering Data

For this research paper, we added filters and constraints on the search terms to allow us to gather and analyze relevant tags, questions and posts. Gathering data from SO was performed

through two primary means: automated data collection using Stack Exchange API and manual inspection through SO posts. A script was written in Python to use the Stack Exchange API v2.3 [20] to query different types of information such as posts, answers, comments, articles, tags, etc. from SO and stored in CSV files. The datasets were gathered in batches per year. This enabled us to answer RQ1.3 about temporal trends in software testing related technology. Each data point can be sorted based on one of the following metrics: activity, relevance, creation, votes, popularity or name.

For RQ1.2&1.3&2, the search terms mentioned in Section 3.1 were used to create the dataset by querying the stack overflow website for posts that contained the search terms either in the title or body of the post. The datasets were queried yearly from 1st Jan to 31st Dec of every year from 2017-2022 and till 31st May 2023 for the final year. The search terms are either mentioned in the title or the body of the post. The following filters used for the queries:

1. Sort - Relevance
2. Order - Descending
3. Starting date - 2017-01-01
4. Ending date - 2017-12-31
5. Title - The search terms provided (system testing, integration testing, etc.)
6. Body - Same search terms provided
7. Additional parameters:
   (a) withbody - To query the content of the post
   (b) has_more - To query the consequent page until the limit of the search result/limit of the API

RQ1.4 was answered by manual inspection of the thread: "What's your most controversial programming opinion?" [21]. Measuring controversy through community-curated indicators (such as accepted answers, and answer vote) are often unreliable [22]. This thread was chosen because it encapsulated the essence of the research question and we could avoid using an arbitrary metric for measuring controversy on SO.

Creation of the dataset for RQ2 was similar to that of RQ1. Along with the filters mentioned, we added an extra filter called "tagged". This filter enabled us to filter posts that were strictly related to software testing. The tags used were different synonyms of test: tests, testing, software-testing. The dataset[2] is publicly available on the 4TU research data website [23].

3.3 Pre-processing the Data

To allow the gathered data to be useful and accurate for subsequent analysis, it is necessary to discard stop-words from the dataset. The stop-words include prepositions, articles, punctuation marks, non-alphabetic characters and error log statements [24]. Search terms were also omitted from the dataset as their presence would dominate the frequency analysis without yielding any useful information. Furthermore, the bodies of the posts contain HTML tags. Similar to search terms, it is necessary to remove them to avoid any skew during analysis. Tags such as , <pre></pre>, <a> and <code></code> were removed.

[2]https://doi.org/10.4121/1e28497e-00d5-4be2-8533-0a143922421c.v1

3.4 Analyzing the Data

Frequency analysis [25] was performed to achieve results for RQ1.2&1.3. We performed the frequency analysis on words by counting the occurrence of words in the text. The definition of a word also includes multiple words combined by a hyphen (-) to enable frequency analysis on tags. The frequency analysis was done for each dataset (per year) and the results were compiled into a bigger dataset. The temporal trends for the popular technology/tool/framework were formulated based on the frequency analysis.

Manual inspection and aggregation of underlying information was the analysis method chosen for RQ1.4&2. We chose not to utilize a data-driven approach because of the limited availability of relevant data. The expected result was a list of best practices mentioned by practitioners. This list is ordered by the occurrence of the practice/advice across different posts, followed by the view count of the post. They were then grouped into broader categories as presented in Section 4.

4 Results

This section contains the results based on the analysis of the datasets and answers to the research questions.

Popular Frameworks in Software Testing

For RQ1, the most discussed frameworks for each type of software testing are summarized in Table 1 and all the frameworks mentioned across the posts are presented in Table 2. These frameworks are derived from the title, body and tags of the posts. Some interesting observations were made during the compilation of this dataset:

- The overlap of frameworks in the tags and contents (title and body) of posts is consistent and indicative of the similarity of the information discussed in the posts and the tags that the post owners assign to the post. This accounts for the credibility of tags on SO.
- The posts available on "spike testing" and "endurance testing" were limited and hence not representative of the frameworks they discussed (they were omitted from the table).
- Selenium, Spring, JMeter and React were mentioned in more than one type of testing category. This indicates their widespread usage across different testing types.

Temporal Trends

While compiling the datasets year by year from 2017-2023, specific patterns and trends appeared from the datasets. These trends concern the different technologies discussed in software testing.

Posts concerning test automation have increased over recent years. This was identified through the increased number of "automated-tests" tags in posts related to software testing. Additionally, the mention of scripts, scripting and continuous

Testing Type	Most discussed Framework
Acceptance	Codeception, Flask
Compatibility	Selenium, Karate
Database	Laravel
End-To-End	Protractor, Cypress
GUI/UI	Espresso
Integration	Spring, .NET, Flutter
Load,Performance, Stress	JMeter, Locust, Gatling
Regression	Selenium,Playwright,Snowflake
Security	Owasp ZAP, Sonarqube
System	Ruby-on-rails
Unit	Angular, Jasmine, React, Spring

Table 1: Summary of most discussed frameworks for different types of software testing

integration/continuous deployment (CI/CD) in posts is an upward trend over recent years. This is tangled with the usage of test management and version control technologies.

Frameworks often see a rise or fall in popularity through time. Documenting such trends gives us a better picture of the frameworks the industry follows. End-to-end testing has seen a switch from Protractor to Cypress. This trend is visualized in Figure 1 [26].

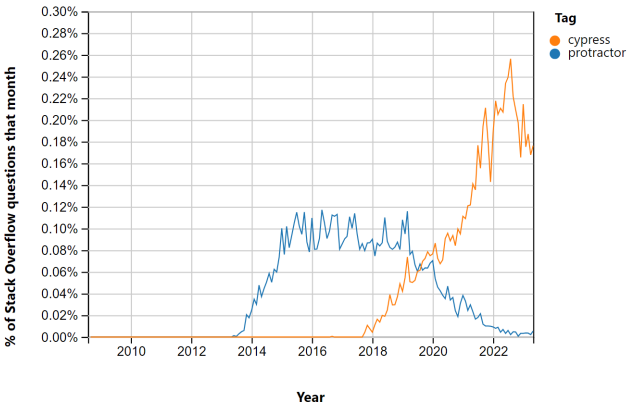


Figure 1: Rise of Cypress vs. Fall of Protractor

Other frameworks have endured the test of time and are predominant in their field. JMeter and Locust for performance testing and Espresso for GUI/UI testing are the epitomai of frameworks that have been constantly popular through the years.

An interesting uptrend of JavaScript frameworks has been noted. NodeJS, JestJS, NestJS and VueJS have proliferated into software testing. This trend is visualized in Figure 2 [26].

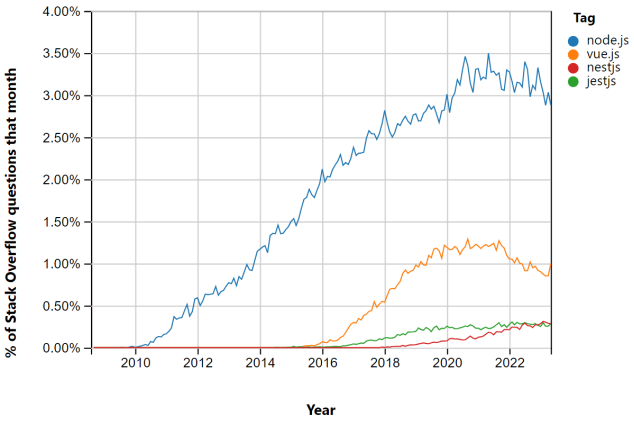
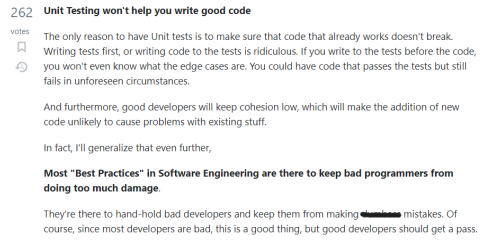


Figure 2: Uptrend of JavaScript Frameworks

Controversial Threads

A few controversial opinions regarding software testing from a SO thread regarding "controversial opinions from programmers" [21] are discussed below. This is a closed thread that has over 400 opinions from practitioners, a majority of which discuss software engineering and computer science topics.

Unit testing is NOT helpful. This thread states that test-driven development (TDD) and unit testing are not good methodologies in software development. The argument given for justifying this opinion was, writing test cases before writing code limits the developer's capacity to identify edge cases that were not identified when writing test cases. The opinion suggests that testing should always be performed after the code is written (Figure 3). Another thread reiterated a similar opinion on unit testing. However, this thread argues that systems are often intertwined and creating loopholes for developing a comprehensive unit testing suite is unnecessary and complex (Figure 4).


262 votes

Unit Testing won't help you write good code

The only reason to have Unit tests is to make sure that code that already works doesn't break. Writing tests first, or writing code to the tests is ridiculous. If you write to the tests before the code, you won't even know what the edge cases are. You could have code that passes the tests but still fails in unforeseen circumstances.

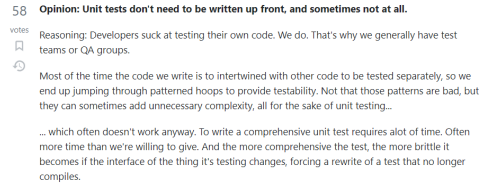
And furthermore, good developers will keep cohesion low, which will make the addition of new code unlikely to cause problems with existing stuff.

In fact, I'll generalize that even further,

Most "Best Practices" in Software Engineering are there to keep bad programmers from doing too much damage.

They're there to hand-hold bad developers and keep them from making [redacted] mistakes. Of course, since most developers are bad, this is a good thing, but good developers should get a pass.

Figure 3: Thread 1 of unit testing related controversial opinion


58 votes

Opinion: Unit tests don't need to be written up front, and sometimes not at all.

Reasoning: Developers suck at testing their own code. We do. That's why we generally have test teams or QA groups.

Most of the time the code we write is to intertwined with other code to be tested separately, so we end up jumping through patterned hoops to provide testability. Not that those patterns are bad, but they can sometimes add unnecessary complexity, all for the sake of unit testing...

... which often doesn't work anyway. To write a comprehensive unit test requires alot of time. Often more time than we're willing to give. And the more comprehensive the test, the more brittle it becomes if the interface of the thing it's testing changes, forcing a rewrite of a test that no longer compiles.

Figure 4: Thread 2 of unit testing related controversial opinion

Developers testing their own code. There are conflicting opinions on whether developers should test the code they write. One thread argues that the communication overhead between developing and testing teams is too high (Fig-

Type of Software Testing	Most occurring technology from post (title and body)	Most occurring technology from tag
Acceptance Testing	Codeception, Flask, Yii, Selenium, Keycloak, Ember, Joomla, Rails, Sonarqube, Mocha, Cucumber, Couchbase, Apache	Codeception, Selenium, Ruby-On-Rails, Cucumber, Couchbase, JestJS, Sonarqube, Magneto, Ember.js, Netty, Behat
Compatibility Testing	Selenium, Openshift, Karate, Kafka, Angular, Firebase	Selenium, Firebase, Karate, Apache-Kafka
Database Testing	Laravel, Kafka, Jest, Selenium, PHPUnit, Spring, JMeter, Cucumber	Laravel, Selenium, Spring-Boot, Cucumber, PHPUnit, JUnit
End-to-end Testing	Protractor, Cypress, Playwright, npm, Angular, Detox, React, Nightwatch, Testcafe, NestJS, Spring, Selenium	Angular, Cypress, Protractor, Playwright, ReactJS, NestJS, JestJS, Puppeteer, Selenium, NodeJS, Detox, React-Native
GUI/UI Testing	Espresso, Android, iOS, Xcode, Xamarin, Swift, Selenium, Karate, React, .NET	Swift, iOS, Selenium, Espresso, Android, Karate, Cucumber
Integration Testing	Spring, .NET, Flutter, JUnit, React, Jest	Spring-Boot, ASP.NET-Core, Flutter, Mockito, JUnit, JestJS, Ruby-on-Rails, NodeJS, Maven, Angular, Netty, ReactJS
Load Testing	JMeter, Locust, Apache, Gatling, Spring	JMeter, Locust, NodeJS, Gatling, Selenium, Spring, Blazemeter, Apache-Kafka
Performance Testing	JMeter, Apache, Kafka, Selenium, Locust, Gatling, Karate, Gradle	JMeter, Gatling, Locust, Selenium, Apache-Kafka, Blazemeter, Karate, Groovy, NodeJS, Spring
Regression Testing	Selenium, React, Playwright, Snowflake, Percy, Angular, Loki, Maven, .NET, Sencha, TestNG	Automated-Tests, Playwright, Android, Selenium, LokiJS, R-Caret, VueJS, BackstopJS, TestNG, Maven, ReactJS, Snowflake, Spring, ASP.Net-Core, Pandas, Puppeteer
Security Testing	JMeter, Owasp ZAP, Selenium, Ajax, Burp, Spring, Sonarqube, Apache, Cucumber, Flutter, ReactJS, JUnit	Owasp ZAP, Sonarqube, Flutter, Spring, Selenium, Jquery, NodeJS, Android, Burp, Laravel
Stress Testing	JMeter, Apache, Cassandra, Spring, ASP.NET, NodeJS, StressNG,	JMeter, NodeJS, Android, ASP.Net-Core, Cassandra, Spring, Apache-Kafka
System Testing	Rails, Mongoose, Spring, Capybara, Harfbuzz, Databricks, Firebird, Gremlin, JUnit	Ruby-on-rails, Capybara, Firebird, Maven, Apache-Spark, NodeJS, Ahoy, Jquery, Pyspark
Unit Testing	Angular, Jasmine, Mock, Jest, React, Spring, Flutter, VueJS	Angular, Karma-Jasmine, Mockito, NodeJS, ReactJS, JestJS, Junit, Spring

Table 2: All testing-related technology mentioned for different types of software testing derived from tags and contents (title and body) of the post.

ure 5) while another argues that development and testing are two different disciplines that follow different principles (Figure 6).

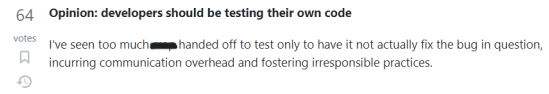


Figure 5: In-favour argument of developers testing their own code

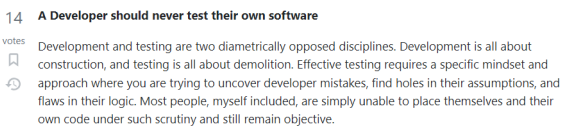


Figure 6: Opposing argument that developers should not test their own code

Recommended Practices

For RQ2, the results for the recommended practices/advice/suggestions that practitioners give to other practitioners are compiled and presented in this section. Many posts emphasised test-driven development (TDD). It was considered an ideal approach towards software development. The recommendations mentioned are in no particular order.

Writing Test Cases

Similar to writing production code, keeping test cases simple allows for easy debugging. Ideally, the test case should not require debugging. The names of test cases and variables used should be indicative of their purpose in the code.

Test cases should serve the purpose of validating the function of the software on valid and invalid inputs. Unit testing with branch coverage often gives a good score for edge cases. Subsequently, pairwise combinatorial testing helps reduce the

number of test cases without compromising the branch coverage.

Avoiding code duplication through refactoring code and extracting common elements from test cases creates an uncomplicated test structure. Positive/negative test cases can be compiled into one test case. It is important to define a test suite architecture for maintenance, debugging, and coherence across the test suite.

Stubbing and mocking are good testing techniques to create data points for edge-case test cases. Constraints in testing data can be introduced using these techniques. Mocking could be a good technique when testing the user interface (UI) and functionality of the system by mocking HTTP requests.

Separate Test Suite and Main Application
Creating a testing environment and separating testing from production is an important step in test suite architecture design. Create separate folders/packages/directories to separate files. Use factories to produce real-time object instances for testing. Common elements from test cases can be refactored as the setup for the test environment.

Test Automation
One of the widely discussed topics is automating test cases for medium to large-scale applications that are developed by a team (not individual). It is essential to discuss the trade-offs in the variables that are chosen for automation. Using tools such as dynamic pipelines and CI/CD (Continuous Integration/Continuous Deployment) are useful tools when automating test cases. Adopting the practice of automating regression tests is very efficient when deploying multiple versions of the application and cautions the creation of new bugs from existing code. A good methodology indicated:

Modify Code → Modify Tests
Add New Code → Add New Tests

How much time should be spent on testing?
It is impossible to receive 100% test coverage and one should not aim to test their code with that motive. Writing tests is useful when looking for bug fixes, understanding the problem and creating a non-flaky solution. About 10-50% of the development time should be spent on testing (depending on the criticality of the system). Software used in high-risk environments such as aeroplanes, nuclear plants or operation rooms must be tested more extensively to reduce the probability of failure.

Testing Databases
When testing databases, it is important to determine the order of tests to avoid mishaps when dealing with data points. Addition of multiple files while testing is acceptable as long as the database is not altered while reacting to added files. It is vital to check the validity and integrity of the database. Selenium provides data-validation options and suggestions.

Importance of log creation during testing
Logs are artifacts created during test execution that contains comprehensive information about the various processes and stages of software execution. Logs are important to identify problems and trace the issue to its source. The usefulness of logs can be attributed to how well it is written and which stage of testing they are used (system-wide integration testing bugs would be easier to trace in comparison to unit testing bugs). Logs can be written in the form of assertions, print statements or log files. They can be helpful in duplicating the scenario that caused the error. Furthermore, log analysis tools (like Logwatch) can help in indicating the status of test runs, how efficient they are, infrastructure impact and display location of errors.

Recommended resources for software testing
Dedicated internet resources that discuss software testing are mentioned in a few posts. Some testing blogs are TestingReflections, Abakas, Satisfice.com and Google Testing Blog. Other active forums that discuss software testing are QAForums, OneStopTesting, SQAForums and SoftwareTestingClub.com.

5 Discussion

This section highlights interesting insights that can be gathered from the data. It also contains a comparison to the past research and limitations faced during the research.

5.1 Interesting Insights

Whilst compiling the list of popular frameworks discussed under different types of software testing, other technologies, techniques and concepts emerged during the frequency analysis.

- Test management and version control technologies such as AWS, Azure, Jenkins, Kubernetes, Docker, Gitlab, and Github were often mentioned. This could show the reliance on such technologies to manage and maintain test suites across projects.
- One of the other popular techniques mentioned across various posts was “mocking”. Mocking is a common technique used throughout all types of software testing. This conclusion is further solidified by the presence of frameworks such as Mockito across different types of software testing.
- A commonly discussed topic that is mentioned in all types of software testing is terminology and debugging. This shows that SO users are confused about nomenclature pertaining to software testing. It could indicate a lack of awareness regarding software testing terminology. Furthermore, users often post screenshots and code snippets of their code asking for help debugging it. This could prove that amateurs do not get enough exposure to testing/debugging and are unaware of how to go about it.

We can also comment on the popularity of unit, integration, load, performance, GUI/UI and end-to-end testing over the other types of testing (Figure 7 omits unit testing because it skews the picture and makes a nuanced comparison of other types of testing harder). It could draw focus on the types of software testing the industry focuses on. The tags on posts concerning one type of testing were often intertwined with another type of testing. This could indicate that types of software testing are not isolated and are often linked to each other.

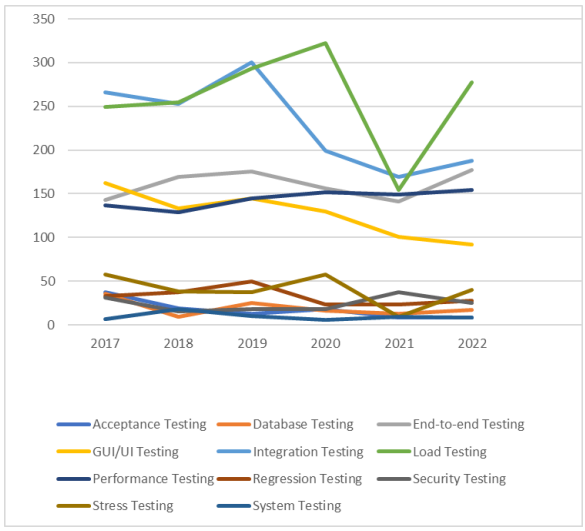


Figure 7: Number of posts per year for different types of software testing

Controversial answers mentioned in the thread often give insights into the complicated nature of the industry [21]. However, it is difficult to measure and model controversy from sources such as SO. The conflicting arguments about whether developers should test their own code show the diverse range of opinions among the software testing community. These opinions may derive from personal experiences and circumstances. It also displays the variety of situations SO users encounter during their work and the absence of a universal principle/strategy that addresses all problems.

5.2 Comparison to past research

Research conducted by Kochhar [6] and Openja et al. [7] are of significant importance in inspiring this research. Their work can be used to compare and draw parallels to the results of this research.

RQ1.2&1.3 concerning popular frameworks and temporal trends were directly inspired by Kochhar's research. Kochhar's research results in discovering testing frameworks and databases as popularly discussed topics from 2009-2014. This paper expands on the previous research and identifies which frameworks are popularly discussed in the realm of software testing. Subsequently, this paper recognizes trends of such frameworks. Both papers share the conclusion that multiple posts concern clarity in terminology. In contrast, this paper adopts frequency analysis as the methodology instead of Latent Dirichlet Allocation (LDA) to recognize the major concept/technology discussed in posts.

The results from Openja et al. are incompatible and incomparable with the ones from this paper due to the difference in the subject (release engineering vs. software testing). However, the research methodology concerning the collection, pre-processing and analysis of the data is inspired by Openja et al.'s research. The analysis of tags is an important technique used in both papers.

5.3 Limitations

A technical limitation faced during this research while using the Stack Exchange API was that it had a limit on the number of requests made daily (10,000). The aggregated data points queried would exceed the limit on some days. In such cases, changing the IP address by moving to a different network or waiting till the next day to query the remainder of the data allowed us to work around this limitation. Additionally, the Stack Exchange API is intentionally limited for querying posts. To prevent crashes and excessive querying, the API implements this policy. To circumvent this, the datasets were queried yearly.

6 Conclusion and Future Work

This paper aims at aggregating software testing knowledge and provide insights from Stack Overflow posts that could be valuable to educators and the computer science community. Previously conducted research from Kochhar and Openja et al. played a major role in shaping the direction of this research. The results of this paper add to the narrative of software testing from 2009-2014. This research opted for automating the querying process from SO using the Stack Exchange API and developed a script for doing so. The data collected [23] was used for answering the following: the popular frameworks for different types of software testing, the temporal trends for various technologies and the best practices discussed among practitioners. The datasets were collected yearly from 2017-2023 using the filters mentioned in Section 3.2. Manual inspection was chosen to identify nuanced controversial opinions on testing from a thread on SO.

The results for the popular frameworks were compiled using frequency analysis of words from the title and contents of the posts and the tags related to the posts. Selenium, Spring, JMeter and React were the commonly mentioned frameworks across all types of testing and they are summarized in Table 1. Automated testing and JavaScript frameworks have seen an uptrend in recent years. Other frameworks have seen a reduction in popularity (Protractor) due to the rise of another (Cypress) in the domain of end-to-end testing. Practitioners provide advice and suggestions on how to go about simple testing activities. This paper aggregated recommendations/best practices mentioned by practitioners on writing test cases, test suite architecture, test automation, database testing, time spent on testing, and log creation during testing.

Test management and version control software and tools have become key components in software testing. Additionally, controversial opinions give an interesting insight into a wide variety of experiences and situations that practitioners draw their opinions from. All these results paint a comprehensive picture of modern practices in software testing from SO. The picture could be skewed because of the knowledge accumulated from a single source (Stack Overflow) or it might not be the full picture (other aspects of software testing may be missed or not discussed). Thus, it is important to mention the potential for further research on this topic.

Approaching the research on software testing using open coding or NLP would reveal other interesting topics that are specifically discussed on SO. The categorization created through this method would determine natural areas of discussion. These topics could be compared with topics taught in higher education institutions to create a qualitative compari-

son between industry discussion and academic syllabus.

Furthermore, collectives are a recent addition to SO. Collectives are dedicated spaces around a specific technology or product (Example: AWS, CI/CD, Google Cloud). Researching them would allow us to gather technology-specific information in the domain of testing.

Measuring controversy is a field of research in itself. A future study of creating a metric for controversy on SO could help gather posts that discuss controversy. Such a study can expand our knowledge of disputed topics in software testing.

7 Responsible Research

SO is a public platform that contains various types of data (questions, answers, opinions, user profiles, etc.). Research using publicly available data must be handled and analyzed carefully. Since they are not facts of the pertaining domain, their presence and inclusion must be rationally argued. The research was conducted responsibly and ethically with pure academic intent. All credits to previous papers and sources used during this research have been referenced. There are aspects of this research that could raise questions about traceability, data trimming and reproducibility. The following section discusses these issues in detail.

Traceability All the data collected are information that is stated or opinionated by SO users. To remove any bias against any group of users, the researcher never queried, collected or stored any information regarding the user. The lack of data collection related to users should allow the users regarding the posts to remain anonymous.

Data Trimming When manually examining a large public dataset such as SO, it might be easy to forget or leave out available information. During this research, the exclusion process of data points was methodical and strict to avoid accidental exclusions. The filters that were applied through the Python script were explicitly documented and reasoned.

Reproducibility The process of conducting this research is well documented in Section 3. The filters used for querying and the original script can be used for reproducing the datasets. The datasets and other metadata used in this research are publicly available on the 4TU website [23].

References

[1] Kinza Yasar. *Software Testing*. Accessed: 06-05-2023. URL: https://www.techtarget.com/whatis/definition/software-testing.

[2] Vahid Garousi et al. "What industry wants from academia in software testing? Hearing practitioners' opinions". In: June 2017. DOI: 10.1145/3084226.3084264.

[3] Stefanie Beyer et al. "What kind of questions do developers ask on Stack Overflow? A comparison of automated approaches to classify posts into question categories." In: vol. 25. 2020, pp. 2258–2301. URL: https://doi.org/10.1007/s10664-019-09758-x.

[4] *Stack overflow*. 2008 Accessed: 16-05-2023. URL: https://stackoverflow.com/.

[5] *2022 Stack Overflow Developer Survey*. Accessed: 21-06-2023. URL: https://survey.stackoverflow.co/2022#developer-profile-education.

[6] Pavneet Singh Kochhar. "Mining Testing Questions on Stack Overflow". In: *Proceedings of the 5th International Workshop on Software Mining*. SoftwareMining 2016. Singapore, Singapore: Association for Computing Machinery, 2016, pp. 32–38. ISBN: 9781450345118. DOI: 10.1145/2975961.2975966.

[7] Moses Openja, Bram Adams, and Foutse Khomh. "Analysis of Modern Release Engineering Topics : – A Large-Scale Study using StackOverflow –". In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2020, pp. 104–114. DOI: 10.1109/ICSME46990.2020.00020.

[8] Stuart Bretschneider, Frederick J. Marc-Aurele Jr., and Jiannan Wu. ""Best Practices" Research: A Methodological Guide for the Perplexed". In: *Journal of Public Administration Research and Theory* 15.2 (Dec. 2004), pp. 307–323. ISSN: 1053-1858. URL: https://doi.org/10.1093/jopart/mui017.

[9] Miltiadis Allamanis and Charles Sutton. "Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code". In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, pp. 53–56. DOI: 10.1109/MSR.2013.6624004.

[10] Ardic Baris and Zaidman Andy. "Hey Teachers, Teach Those Kids Some Software Testing". In: (2023). To appear in: IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG).

[11] *Stack Overflow Traffic*. Accessed: 17-06-2023. URL: https://stackexchange.com/sites#traffic.

[12] Sebastian Baltes et al. "SOTorrent: reconstructing and analyzing the evolution of stack overflow posts". In: *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*. Ed. by Andy Zaidman, Yasutaka Kamei, and Emily Hill. ACM, 2018, pp. 319–330. DOI: 10.1145/3196398.3196430.

[13] Sita Shreeraman. *ISTQB – Different Test Types*. Accessed: 20-06-2023. URL: https://www.getsoftwareservice.com/different-test-types/.

[14] Roshali Silva et al. "Effective use of test types for software development". In: *17th International Conference on Advances in ICT for Emerging Regions, ICTer 2017 - Proceedings*. Vol. 2018. 2017, pp. 7–12. DOI: 10.1109/ICTER.2017.8257795.

[15] Harrine Freeman. "Software Testing". In: vol. 5. 3. 2002, pp. 48–50. DOI: 10.1109/MIM.2002.1028373.

[16] Pittet Sten. *Atlassian: The different types of software testing*. Accessed: 26-05-2023. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

[17] Leseu Yan. *SENLA: The Main Types of Software Testing Methodologies and Testing Based on Business Objectives*. Accessed: 26-05-2023. 2021. URL: https://senlainc.com/blog/the-main-types-of-software-testing-methodologies-and-testing-based-on-business-objectives/.

[18] Thomas Hamilton. *Types of Software Testing (100 Examples)*. Accessed: 20-06-2023. 2023. URL: https://www.guru99.com/types-of-software-testing.html.

[19] Stefanie Beyer and Martin Pinzger. "Synonym Suggestion for Tags on Stack Overflow". In: May 2015, pp. 94–103. DOI: 10.1109/ICPC.2015.18.

[20] *Stack Exchange API v2.3 Documentation*. Accessed: 13-06-2023. URL: https://api.stackexchange.com/.

[21] *What's your most controversial programming opinion?* Accessed: 19-06-2023. URL: https://stackoverflow.com/questions/406760/whats-your-most-controversial-programming-opinion.

[22] Xiaoxue Ren et al. "Discovering, Explaining and Summarizing Controversial Discussions in Community QA Sites". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019, pp. 151–162. DOI: 10.1109/ASE.2019.00024.

[23] Dibyendu Gupta. *Data underlying the publication: Mining Software Testing Knowledge from Stack Overflow*. 2023. DOI: https://doi.org/10.4121/1e28497e-00d5-4be2-8533-0a143922421c.v1.

[24] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. "What are developers talking about? An analysis of topics and trends in Stack Overflow". In: vol. 19. 2014, pp. 619–654. URL: https://doi.org/10.1007/s10664-012-9231-y.

[25] 101computing.net. *Frequency Analysis*. Accessed: 20-06-2023. 2009. URL: https://www.101computing.net/frequency-analysis/.

[26] *Stack Overflow Insights - Trends*. Accessed: 10-06-2023. URL: https://insights.stackoverflow.com/trends?tags=.