What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow

Syed Ahmed
Oakland University, USA
sfahmed@oakland.edu

Mehdi Bagherzadeh
Oakland University, USA
mbagherzadeh@oakland.edu

ABSTRACT

Background Software developers are increasingly required to write concurrent code. However, most developers find concurrent programming difficult. To better help developers, it is imperative to understand their interest and difficulties in terms of concurrency topics they encounter often when writing concurrent code.

Aims In this work, we conduct a large-scale study on the textual content of the entirety of Stack Overflow to understand the interests and difficulties of concurrency developers.

Method First, we develop a set of concurrency tags to extract concurrency questions that developers ask. Second, we use latent Dirichlet allocation (LDA) topic modeling and an open card sort to manually determine the topics of these questions. Third, we construct a topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories. Fourth, we investigate the coincidence of our concurrency topics with findings of previous work. Fifth, we measure the popularity and difficulty of our concurrency topics and analyze their correlation. Finally, we discuss the implications of our findings.

Results A few findings of our study are the following. *(1)* Developers ask questions about a broad spectrum of concurrency topics ranging from *multithreading* to *parallel computing*, *mobile concurrency* to *web concurrency* and *memory consistency* to *runtime speedup*. *(2)* These questions can be grouped into a hierarchy with eight major categories: *concurrency models*, *programming paradigms*, *correctness*, *debugging*, *basic concepts*, *persistence*, *performance* and *GUI*. *(3)* Developers ask more about *correctness* of their concurrent programs than *performance*. *(4)* Concurrency questions about *thread safety* and *database management systems* are among the most popular and the most difficult, respectively. *(5)* Difficulty and popularity of concurrency topics are negatively correlated.

Conclusions The results of our study can not only help concurrency developers but also concurrency educators and researchers to better decide where to focus their efforts, by trading off one concurrency topic against another.

CCS CONCEPTS

• **General and reference → Empirical studies**; • **Theory of computation → Concurrency**;

ESEM '18, October 11–12, 2018, Oulu, Finland

ACM ISBN 978-1-4503-5823-1/18/10…$15.00
https://doi.org/10.1145/3239235.3239524

KEYWORDS

Concurrency topics, concurrency topic hierarchy, concurrency topic difficulty, concurrency topic popularity, Stack Overflow

ACM Reference Format:
Syed Ahmed and Mehdi Bagherzadeh. 2018. What Do Concurrency Developers Ask About? A Large-scale Study Using Stack Overflow . In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '18), October 11–12, 2018, Oulu, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3239235.3239524

1 INTRODUCTION

Software developers are increasingly required to write concurrent code to satisfy both functional and nonfunctional requirements of their software. For example, software with a graphical user interface must be concurrent in order to satisfy a functional requirement that the software should be able to display more than one window at a time. Similarly, software with performance requirements must be concurrent in order to satisfy a nonfunctional requirement that the software should support better performance. Software is concurrent if its different computations can potentially run at the same time and otherwise is sequential. However, most developers think sequentially [19] and find concurrent programming difficult.

To better help developers with concurrent programming, it is imperative to understand their interests and difficulties in terms of the concurrency topics they encounter often when writing concurrent code and their difficulties when working with these topics. Such understanding not only can help concurrency developers but also education, development and research communities that support these developers to better decide when and where to focus their efforts [2, 5, 7, 23, 24, 28, 31, 34]. Without such understanding, developers may not prepare themselves for similar difficulties, educators may develop the wrong educational material and researchers may make incorrect assumptions.

With more than three million developer participants, thirty eight million question and answer posts in two billion words, Stack Overflow [30] has become a large and popular knowledge repository for developers to post questions, receive answers and learn about a broad range of topics. This makes Stack Overflow a great source to learn about developers' interests and difficulties [5, 7, 24, 25, 28, 34].

To understand interests and difficulties of concurrency developers, we conduct a large-scale study on the textual content of the entirety of Stack Overflow to answer these research questions:

- ***RQ1. Concurrency topics*** What concurrency topics do developers ask questions about?
- ***RQ2. Topic hierarchy*** What categories do these concurrency topics belong to? What does the hierarchy of these concurrency topics look like?

- ***RQ3. Popularity*** What topics are more popular among concurrency developers?
- ***RQ4. Difficulty*** What topics are more difficult to successfully find answers to their questions?
- ***RQ5. Correlation*** What popular concurrency topics are more difficult? How do popularity and difficulty of concurrency topics correlate?

To answer these questions, we take the following major steps. First, we develop a set of concurrency tags to identify and extract concurrency questions that developers ask on Stack Overflow. Second, we use latent Dirichlet allocation (LDA) [9] topic modeling and an open card sort [11] to manually determine the topics of these questions using their textual contents. Third, we construct a topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories. Fourth, we investigate the coincidence of our concurrency topics with findings of previous work. Fifth, we measure the popularity and difficulty of our concurrency topics using several well-known metrics used by previous work [5, 7, 22, 28, 28, 31, 34] and analyze their correlation. Finally, we discuss the implications of our findings for concurrency developers, educators and researchers.

A few findings of our study are the following. ***Concurrency topics:*** *(1)* Developers ask questions about a broad spectrum of concurrency topics ranging from *multithreading* to *parallel computing*, *mobile concurrency* to *web concurrency* and *memory consistency* to *runtime speedup*. ***Topic hierarchy:*** *(2)* These questions can be grouped into a hierarchy with eight major categories: *concurrency models*, *programming paradigms*, *correctness*, *debugging*, *basic concepts*, *persistence*, *performance* and *GUI*. *(3)* Developers ask more about *correctness* of their concurrent programs than *performance*. *(4)* Our concurrency topics and categories coincide with several findings of previous work by Pinto et al. [25], Barua et al. [7], Rosen and Shihab [28] and Lu et al. [19], among others. ***Popularity & difficulty and their correlation:*** *(5)* Concurrency questions about *thread safety* and *database management systems* are among the most popular and the most difficult, respectively. *(6)* Difficulty and popularity of concurrency topics are negatively correlated.

Our dataset is available at https://goo.gl/uYCQPU.

2 METHODOLOGY

Figure 1 shows an overview of the methodology used to study interests and difficulties of concurrency developers on Stack Overflow.

Step ❶: Download Stack Overflow dataset In the first step of our analysis, we download the Stack Overflow dataset which is publicly available through Stack Exchange Data Dump [29]. The dump includes a large set $\mathcal{S}$ of question and answer posts with a set of data for each post. Among others, the data for a post includes its identifier, its type (question or answer), title, body, tags, creation date, view count, score, favorite count and the identifier of the accepted answer for the post if the post is a question. An answer to a question is accepted if the contributor who posted the question marks it as accepted. Our dataset includes 38,485,046 questions and answers posted over a time span of over 9 years from August 2008 to partway through December 2017 by 3,589,412 developer participants of Stack Overflow. Among these posts 14,995,834 (39%) are questions and 23,489,212 (61%) are answers of which 8,034,235 (21%) are marked as accepted answers.

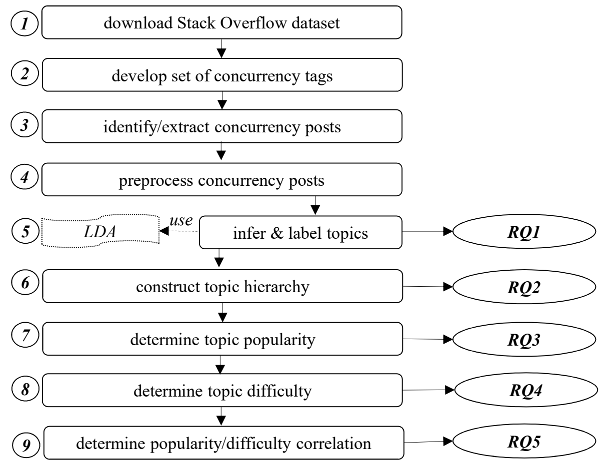

Figure 1: Overview of our methodology to study interest and difficulties of concurrency developers using Stack Overflow.

Step ❷: Develop concurrency tag set We consider a post as a concurrency post if it has a concurrency tag. To develop a set of concurrency tags we take the following steps. First, we manually inspect Stack Overflow's top 100 most used tags and select its concurrency-related tags to form the initial set $\mathcal{T}_0$ of concurrency tags. $\mathcal{T}_0$ includes a single tag "multithreading" [19, 25]. Second, we go through the Stack Overflow dataset $\mathcal{S}$ and extract questions $\mathcal{P}$ whose tags contain a tag from $\mathcal{T}_0$. The set $\mathcal{P}$ includes 103,747 questions. Third, we extract tags of the posts in $\mathcal{P}$ to form the set of candidate concurrency tags $\mathcal{T}$. Finally, we use two heuristics α and β to refine $\mathcal{T}$ by keeping tags that are significantly relevant to concurrency and excluding others. The heuristic α measures the relevance of a tag t in $\mathcal{T}$ to concurrency.

$$\alpha = \frac{\text{number of posts with tag } t \text{ in } \mathcal{P}}{\text{number of posts with tag } t \text{ in } \mathcal{S}}$$

Similarly, the heuristic β measures the significance of a tag t in $\mathcal{T}$.

$$\beta = \frac{\text{number of posts with tag } t \text{ in } \mathcal{P}}{\text{number of posts in } \mathcal{P}}$$

We consider a tag t to be significantly relevant to concurrency if its α and β values are higher than or equal to specific thresholds. Our experiments using a broad range of thresholds for α and β show that $\alpha = 0.1$ and $\beta = 0.01$ allows for a significantly relevant set of concurrency tags. With these threshold values the set $\mathcal{T}$ of our concurrency tags becomes

$\mathcal{T}$= { *concurrency locking multiprocessing multithreading mutex parallel-processing pthreads python-multithreading synchronization task-parallel-library thread-safety threadpool* }

Note that after refinement, the concurrency tag set $\mathcal{T}$ includes tags like "concurrency". Also, using $\mathcal{T}$ to identify concurrency posts does not prevent a concurrency post to also have tags, such as "asynchronous", that are not explicitly in $\mathcal{T}$[1]. Our threshold values are consistent in range with thresholds used by previous work

[1] To illustrate, our set $\mathcal{C}$ of concurrency posts, with tags $\mathcal{T}$, includes questions with "asynchronous" tag. In addition, topics such as *task parallelism* and *mobile concurrency* in Table 1 cover posts related to asynchrony.

Table 1: Concurrency tags for select relevance and significance threshold values. Our concurrency tag set $\mathcal{T}$ is in gray.

(α, β)	*Set of tags*	*No.*
(0.3, 0.015)	concurrency multithreading pthreads thread-safety threadpool	5
(0.3, 0.01)	concurrency multithreading mutex pthreads python-multithreading thread-safety threadpool	7
(0.3, 0.005)	backgroundworker concurrency executorservice multithreading mutex pthreads python-multithreading runnable semaphore synchronized thread-safety threadpool	12
(0.2, 0.015)	concurrency locking multithreading pthreads synchronization thread-safety threadpool	7
(0.2, 0.01)	concurrency locking multithreading mutex pthreads python-multithreading synchronization task-parallel-library thread-safety threadpool	10
(0.2, 0.005)	atomic backgroundworker concurrency deadlock executorservice grand-central-dispatch locking multithreading mutex openmp pthreads python-multithreading runnable semaphore synchronization synchronized task-parallel-library thread-safety threadpool wait	20
(0.1,0.015)	concurrency locking multithreading parallel-processing pthreads synchronization thread-safety threadpool	8
(0.1, 0.01)	concurrency locking multiprocessing multithreading mutex parallel-processing pthreads python-multithreading synchronization task-parallel-library thread-safety threadpool	12
(0.1, 0.005)	atomic backgroundworker concurrency deadlock executorservice grand-central-dispatch locking multiprocessing multithreading mutex openmp parallel-processing pthreads python-multithreading queue runnable semaphore synchronization synchronized task task-parallel-library thread-safety threadpool wait	24
(0.05, 0.015)	asynchronous c++11 concurrency locking multithreading parallel-processing pthreads sockets synchronization thread-safety threadpool	11
(0.05, 0.01)	android-asynctask asynchronous boost c++11 concurrency locking multiprocessing multithreading mutex parallel-processing pthreads python-multithreading sockets synchronization task-parallel-library thread-safety threadpool timer	18
(0.05, 0.005)	android-asynctask async-await asynchronous atomic backgroundworker boost c++11 concurrency deadlock executorservice grand-central-dispatch locking multiprocessing multithreading mutex openmp parallel-processing process pthreads python-multithreading queue runnable semaphore sockets synchronization synchronized task task-parallel-library thread-safety threadpool timer wait	32

[7, 34]. Rosen and Shihab [28] and Yang et al. [34] use similar approaches to develop their set of security and mobile tags. Table 1 shows the set of concurrency tags for a select set of threshold values for α and β with $\mathcal{T}$ in gray.

Step ❸: Extract concurrency posts After developing the set of concurrency tags $\mathcal{T}$, we extract Stack Overflow posts whose tag set contains a tag in $\mathcal{T}$. This set includes 156,777 question and 249,662 answer posts of which 88,764 (36%) are accepted answers. To reduce noise, following previous work [7, 28], we add questions and their accepted answers from this set to the set of concurrency post $\mathcal{C}$ and discard unaccepted answers. $\mathcal{C}$ includes 156,777 questions and 88,764 accepted answers, i.e. 245,541 posts in total.

Step ❹: Preprocess concurrency posts We preprocess the set of concurrency posts $\mathcal{C}$ to reduce the noise [7, 34] for the next steps of the analysis, by taking the following actions. First, we remove code snippets, enclosed in $\langle code \rangle$, HTML tags, such as $\langle p \rangle$ and $\langle /p \rangle$, stop words, such as "a", "the" and "is" and numbers, punctuation marks, non alphabetical characters and URLs. Second, we reduce words to their base representations. For example "reading", "read" and "reads" all reduce to their base "read". For stop words and word reduction we use MALLET's [27] list of stop words and Porter stemming algorithm [27], respectively.

Step ❺: Infer and label topics After preprocessing, we use latent Dirichlet allocation (LDA) [9] to automatically infer topics through an unsupervised topic modeling of textual contents of concurrency posts $\mathcal{C}$. In a topic model, a document is a probabilistic distribution of topics where the topic itself is a probabilistic distribution of words. A topic is a set of frequently co-occurring words that approximates a real-world concept. In our analysis, a document is a question or an answer post. A document can have multiple topics that cover various proportions of the document and a topic can span over multiple documents. To illustrate, the set of co-occurring words $\{task, execute, async, complete, run, cancel, wait, asynchronous, parallel, schedule\}$ is a topic that approximates the "task parallelism" concept. Task parallelism is a concurrent programming model that allows partitioning a computation into tasks, assigning tasks to concurrently running threads/processes for execution and collecting tasks' results with their completion.

Several implementations of LDA are available. We use MALLET [21] to train a topic model with K topics and I iterations using standard values $50/K$ and 0.01 for MALLET's hyperparameters, following previous work [5, 7, 8, 28, 34]. Our experiments with a broad range of values show that $K = 30$ and $I = 1,000$ allows for the inference of sufficiently granular topics. To produce the model, we treat each individual question post and accepted answer post as an individual document. MALLET processes 245,541 documents.

Topic inference produces a set of words as topics and their proportions. However, the inference cannot automatically assign name labels to these topics. Following previous work [5, 7, 22, 28, 34], we use an open card sort [11] to assign a label to a topic word set w by manually inspecting the top 20 words in w and reading through 15 random posts with w as their dominant topic word set. A topic is the dominant topic of a document if the proportion of the document that the topic covers is higher than proportions that other topics of the document cover. In an open sort, the sorting begins with no predefined categories and participants develop their own categories. The two authors individually assigned topics to word sets, reiterated and refined topics as necessary, and then mutually agreed on a final set of topics. The second author is a Programming Languages professor with extensive expertise in concurrent and event-based systems and the first author is a graduate student with coursework in concurrent and distributed systems. During topic labeling, we merged two pairs of topics because they had sufficiently similar keywords and questions. Pairs are merged into *object-oriented concurrency* and *basic concepts* topics. We removed one unrelated topic that is about synchronization between local and remote repositories in version control systems such as Git and is not about concurrency. The initial level of inter-rater agreement,

Table 2: Names, categories (separated by \) and top 10 words (stemmed) for our concurrency topics of Stack Overflow.

No.	Topic name	Category	Topic words
1	basic concepts	basic concepts	code question work answer understand read find edit issu make
2	task parallelism	concurrency models	task execut async complet run cancel wait asynchron parallel schedul
3	producer consumer concurrency	concurrency models	queue messag consum produc process item buffer block wait empti
4	parallel computing	concurrency models	parallel node loop comput calcul openmp result algorithm mpi function
5	process life cycle management	multiprocessing\concurrency models	process child parent termin exit fork creat kill share start
6	python multiprocessing	multiprocessing\concurrency models	python script run multiprocess process function modul command parallel php
7	thread life cycle management	multithreading\concurrency models	thread main creat run start execut background separ join finish
8	thread sharing	multithreading\concurrency models	function variabl pass call pointer pthread argument type return global
9	thread scheduling	multithreading\concurrency models	loop time wait stop run start sleep set check finish
10	thread pool	multithreading\concurrency models	worker pool job number work process task creat limit threadpool
11	concurrent collections	correctness	list arrai map element collect iter number item kei add
12	thread safety	correctness	thread java safe multipl multi multithread concurr singl implement applic
13	locking	correctness	lock mutex wait condit releas semaphor acquir deadlock synchron resourc
14	memory consistency	correctness	read memori write variabl oper atom cach share synchron access
15	entity management	persistence	concurr session spring entiti transact actor collect updat model version
16	database management systems	persistence	databas tabl queri updat row record lock insert sql transact
17	file management	persistence	file read write log line open stream directori folder text
18	object-oriented concurrency	programming paradigms	object class method instanc creat static access refer variabl synchron
19	web concurrency	programming paradigms	request servic web server applic user app net respons http
20	event-based concurrency	programming paradigms	event signal handler timer callback call handl fire receiv slot
21	mobile concurrency	programming paradigms	android imag app activ game view updat frame asynctask devic
22	client-server concurrency	programming paradigms	server client connect send socket messag receiv data port read
23	data scraping	performance	data time problem load download work page structur solut url
24	runtime speedup	performance	time core cpu run perform memori number system process machin
25	unexpected output	debugging	code program work run output problem print line result function
26	irreproducible behavior	debugging	error test code run problem applic issu window compil crash
27	GUI	GUI	updat form gui window applic button control user progress click

before reiterations and refinements, using Cohen's kappa score was 0.659 (moderate agreement).

Table 2 shows the inferred top 10 words that describe a topic, in the column topic words, and its manually assigned topic name, in the column topic name, for each topic of our 27 concurrency topics. This table shows stemmed topic words that are reduced to their base using the Porter stemming algorithm [27].

Step ❻: Construct topic hierarchy We construct the topic hierarchy by repeated grouping of similar topics into categories and lower level categories into higher level categories.

Table 2 and Figure 3 show the textual and pictorial representations of the topic hierarchy. To illustrate, *thread life cycle management* and *thread scheduling* topics are grouped into a lower level category called *multithreading* where *multithreading* itself is grouped into a higher level category called *concurrency models*. The higher level category *concurrency models* includes other categories such as *multiprocessing* and *parallel computing*.

Step ❼: Determine topic popularity We measure the popularity of a concurrency topic using three metrics, used by previous work. The first metric is the average number of views for questions with the topic as their dominant topic [5, 22, 28, 34]. This metric includes views by both registered users and visitors of Stack Overflow. The inclusion of visitors’ views is important because in Stack Overflow there are many more visitors than there are registered users [20]. The second metric is the average number of questions of the topic marked as favorite by users [5, 22, 25, 34]. The third metric is the average score of questions of the topic [5, 22, 25, 34]. Intuitively, a topic with higher number of views and favorites and a higher score is more popular.

Table 3 shows popularity measurements of concurrency topics.

Step ❽: Determine topic difficulty We measure the difficulty of a concurrency topic using two metrics, used by previous work. The first metric is the percentage of questions of the topic that have no accepted answers [28, 31, 34]. And the second metric is the average median time needed for a question to receive an accepted answer [28, 34]. Intuitively, a topic with higher chance of its questions not receiving accepted answers or taking longer to receive accepted answers is more difficult.

Table 4 shows difficulty measurements of concurrency topics.

Step ❾: Determine correlations After determining popularity and difficulty, we use Kendall correlation tests to identify correlations, if any, between the three popularity metrics and two difficulty metrics of our concurrency topics.

In this paper, the popularity and difficulty of our concurrency topics are bound by their corresponding metrics defined in steps 7 and 8, which are different from other notions of difficulty [15].

3 RESULTS

In this section, we present and discuss the results of our study for research questions ***RQ1-RQ5***. We also investigate the coincidence of our results with findings of relevant previous works.

3.1 RQ1: Concurrency Topics

Topics of concurrency questions that developers ask on Stack Overflow are determined using LDA topic inference and topic labeling, as discussed in Section 2. Table 2 shows these concurrency topics.

As Table 2 shows, developers ask questions about a broad spectrum of concurrency topics ranging from *thread pool* to *parallel computing*, *mobile concurrency* to *web concurrency* and *memory consistency* to *runtime speedup*.

The meaning of these concurrency topics may be best understood by looking at questions that developers ask about in each of these topics. To illustrate, the following is a question in the *thread pool* topic in which the developer is asking how to implement a thread pool where the size of the thread pool can change based on its number of jobs. The Stack Overflow identifier for this question

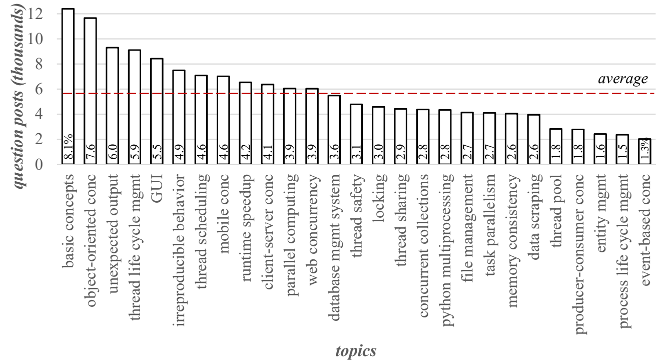

Figure 2: Concurrency topics with individual numbers, average number (dashed line) and percentages of their questions.

is *11249342* and it can be accessed at https://stackoverflow.com/questions/11249342.

> Q.11249342 ***Creating a dynamic (growing/shrinking) thread pool*** *I need to implement a thread pool in Java (java.util.concurrent) whose number of threads is at some minimum value when idle, grows up to an upper bound (but never further) when jobs are submitted into it faster than they finish executing, and shrinks back to the lower bound when all jobs are done . . . How would you implement something like that? ..*

Similarly, the following is a question in the *web concurrency* topic in which the developer is asking how to send emails using background threads in their web application where the background thread prevents blocking of the main thread and therefore does not force the user to wait until the email is sent. Classic ASP is a scripting language to write server side web applications.

> Q.17052243 ***How to perform multithreading/background process in classic ASP*** *I need to send emails via a background job on a classic-ASP app so the user doesn’t have to wait for a slow webserver to complete sending the email. I know I can use Ajax to generate two separate requests, but I’d rather not require Javascript. Plus, I suspect there’s a better way to pull this off. Ideas?*

Finally, the following is a question in *basic concepts* topic that asks about basic motivations behind the need for concurrency.

> Q.541344 ***What challenges promote the use of parallel/concurrent architectures?*** *However, I am so used to thinking about solutions in a linear/serial/OOP/functional way that I am struggling to cast any of my domain problems in a way that merits using concurrency.*

Before proceeding to the next research question, we examine the number of questions that developers ask for concurrency topics. Figure 2 shows the number of questions for these topics and their percentages. As Figure 2 shows, the numbers of questions that developers ask in different concurrency topics are not uniform. Developers ask the most number of questions (8%) about *basic concepts* which is inline with the general understanding that concurrency remains difficult for developers and they still ask questions about its basics. Developers ask the least number of questions (1%) about *event-based concurrency*.

Pinto et al. [25] study the 250 most popular concurrency questions on Stack Overflow. Our observation that the concurrency

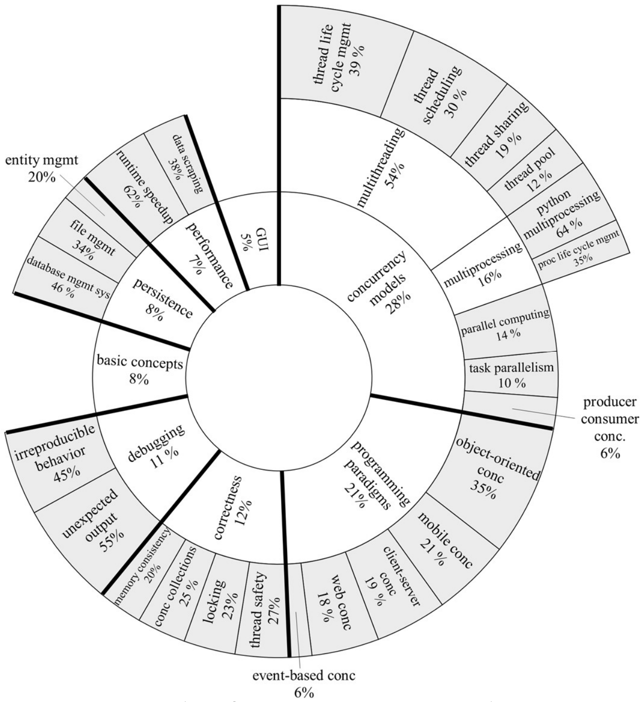

Figure 3: Hierarchy of concurrency topics with concurrency topics in gray and their categories in white.

topic *basic concepts* has the most number of questions coincides with Pinto et al.'s observation that "most of them [questions] are related to basic concepts".

> ***Finding 1:*** Developers ask questions about a broad spectrum of concurrency topics ranging from *thread pool* to *parallel computing*, *mobile concurrency* to *web concurrency* and *memory consistency* to *runtime speedup*.

> ***Finding 2:*** Developers ask the most (8%) about *basic concepts*, inline with the general understanding that concurrency remains difficult for developers and they still ask questions about its basics.

3.2 RQ2: Topic Hierarchy

A hierarchy for concurrency topics that developers ask questions about on Stack Overflow is constructed by repeated grouping of similar topics into categories and lower level categories into higher level categories, as described previously.

Figure 3 shows the hierarchy of concurrency topics with concurrency topics in gray and concurrency categories in white. The inner levels of the hierarchy are its higher levels and the hierarchy expands outwards to lower level categories and concurrency topics at the outermost level. Figure 3 also shows percentages for number of questions a lower level topic/category contributes to its higher level category.

As Figure 3 shows, the questions that developers ask about concurrency can be grouped into a hierarchy with eight high level categories: *concurrency models*, *programming paradigms*, *correctness*, *debugging*, *basic concepts*, *persistence*, *performance* and *GUI*. In addition, the number of questions that developers ask in each category is not uniform. Developers ask the most questions (28%) about the *concurrency models* category and the least (5%) about *GUI*. In addition, developers ask more questions (12%) about *correctness* of their concurrent programs than their *performance* (7%). This is inline with the general tradeoff between performance advantages of concurrency and its correctness issues [3, 4, 16, 18].

> ***Finding 3:*** Questions that developers ask about concurrency can be grouped into eight major categories: *concurrency models*, *programming paradigms*, *correctness debugging*, *basic concepts*, *persistence*, *performance* and *GUI*.

> ***Finding 4:*** Developers ask the most (28%) about *concurrency models* and the least (5%) about *GUI*.

> ***Finding 5:*** Developers ask more (12%) about concurrency *correctness* than *performance* (7%), inline with the tradeoff between concurrency's performance benefits and correctness issues.

We continue this section by a detailed discussion of concurrency categories and their constituent topics and their coincidence with relevant previous works.

3.2.1 Concurrency models. Concurrency models are mainly concerned about concurrency abstractions (e.g. threads and processes) and execution models (e.g. multicore and single core executions). The *concurrency models* category includes five lower level categories among which *multithreading* alone contains more than half of the questions that developers ask in this category. This is inline with the general understanding that multithreading is the defacto dominant concurrency model. Developers ask the most (54%) about *multithreading* and the least (6%) about *producer consumer concurrency* when asking questions about concurrency models.

In *multithreading*, developers ask questions with titles like (thread life cycle management): *"How to safely destruct Posix thread pool in C++?"* and (thread pool): *"Is there a way to create a pool of pools using the Python workerpool module?"*. Whereas *producer consumer concurrency* includes question with titles like *"producer Consumer with BlockingQueues in Java EE as background task"*. The name inside parentheses is the concurrency topic of the question.

Pinto et al. [25] categorize the 250 most popular concurrency questions into several concurrency themes. Our *multithreading* category and its *thread life cycle management* topic coincide with their *threading* and *thread life cycle* themes. Rosen and Shihab [28] study and categorize Stack Overflow questions related to mobile development into several mobile topics including *threading*. Our *multithreading* category coincides with their mobile *threading* topic.

> ***Finding 6:*** Developers ask the most (54%) about *multithreading* when asking about concurrency models, inline with the general understanding that multithreading is the defacto dominant concurrency model. Developers ask the least (6%) about *producer consumer concurrency*.

3.2.2 Programming paradigms. Programming paradigms are mostly concerned about programming abstractions (e.g. objects and events), platforms (e.g. web and mobile) and patterns (e.g. producer consumer). The *programming paradigms* category includes five lower level categories among which *object-oriented concurrency* alone contains more than a third of questions that developers ask in this category. This is inline with the general understanding that object-orientation is a dominant programming paradigm. Developers ask the most (35%) about *object-oriented concurrency* and the least (6%) about *event-based concurrency* when asking about programming paradigms.

In the *object-oriented concurrency* and *event-based concurrency* topics developers ask questions with titles like *"Is it better to synchronize object from inside of the class that encapsulates access it or from outside?"* and *"What type of timer event should I use for a background process when my timer fires very quickly?"*.

Barua et al. [7] study and categorize all Stack Overflow questions and answers into several general topics including web and mobile development. Our concurrency topics *mobile concurrency* and *web concurrency* coincide with their general web development and mobile development topics. Our *mobile concurrency* topic is inline with Pinto et al.'s [25] observation that "concurrent programming has reached mobile developers". Out of the 250 most popular concurrency questions in their study, 22 are related to mobile development.

> ***Finding 7:*** Developers ask the most (35%) about *object-oriented concurrency* when asking about concurrent programming paradigms, inline with the general understanding that object-orientation is a dominant programming paradigm for concurrency. Developers, ask the least (6%) about *event-based concurrency.*

3.2.3 Correctness. Correctness is concerned with prevention of data corruption for concurrently accessed (e.g. read and write) data using mechanisms like locking, consistent memory models and thread safe data structures and programming patterns. *Correctness* questions are almost evenly divided among its topics *thread safety*, *locking*, *concurrent collections* and *memory consistency.*

In the *correctness* category developers ask questions with titles like (thread safety): *"How to make factory [pattern] thread safe?"* (locking): *"Is there a way to lock 2 or more locks or monitors atomically?"*, (concurrent collections): *"Threadsafe dictionary that does lookups with minimal locking"* and (memory consistency): *"Atomic read-modify-write in C#"*.

Lu et al. [19] categorize concurrency bug patterns and their fixes. Our *memory consistency* topic coincides with their observations that most concurrency bugs are atomicity violation bugs where the "desired serializability among multiple memory accesses is violated" and order violation bugs where the "desired order between two (groups of) memory accesses is flipped." Our *locking* topic coincides with their designation that locking is one of the main fixes for concurrency bugs to ensure correctness. In addition, our *correctness* category and its *locking* topic coincide with Pinto et al.'s [25] correctness and locking themes. Similarly, our *concurrent collections* topic coincides with their concurrent libraries theme.

> ***Finding 8:*** Developers ask almost equally about *thread safety* (27%), *locking* (25%), *concurrent collections* (23%) and *memory consistency* (20%) when asking about correctness.

3.2.4 Basic concepts. Basic concepts include questions about both theoretical and practical questions about concurrency with titles like *"How many threads are involved in deadlock?"*, *"What is a race condition?"*, *"Lock, mutex, semaphore... what's the difference?"* and *"Java: notify() vs. notifyAll() all over again"*.

Our *basic concepts* category coincides with Pinto et al.'s [25] themes for theoretical concepts and practical concepts.

3.2.5 Debugging. Debugging is mainly concerned about finding and fixing concurrency bugs which manifest either in the behavior or output of programs. Debugging questions are almost evenly divided among its topics *irreproducible behavior* and *unexpected output* which includes question with titles like *"Trace non-reproducible bug in C++"* and *"Synchronized codes with unexpected outputs"*.

Our *irreproducible behavior* topic coincides with Lu et al.'s [19] observation that some "concurrency bugs are very difficult to repeat".

3.2.6 Persistence. Persistence is about storing and retrieving of data using persistence management systems (e.g. databases management systems, entity/object persistence[2] or file systems). *Persistence* includes three topics among which *database management systems* includes near half of persistence questions. The *persistence* category includes question with titles like (database management system): *"How do I lock read/write to MySQL tables so that I can select and then insert without other programs reading/writing to the database?"*, (file management): *"Is this is correct use of mutex to avoid concurrent modification to file?"*, and (entity management): *"Save entity using threads with JPA [Java Persistence API] (synchronized)"*.

Our *database management systems* topic coincides with Barua et al.'s [7] general *MySQL* topic.

3.2.7 Performance. Performance is about speeding up execution of programs (e.g. data scraping programs[3]).

Performance includes two topics with question with titles like (runtime speedup): *"Poor multithreading performance in .Net"* and (data scraping): *"Using multithreading to speed up web crawler written by beautifulsoup4 and python"*.

Interestingly, Pinto et al. [25] mentions that they "did not find questions that ask for advices on how to use concurrent programming constructs to improve application performance, which is surprising, since performance is one of the most important motivations for the use of concurrency and parallelism". In contrast our *performance* topic includes more than 7% of all concurrency questions.

3.2.8 GUI. Graphical user interface allows for the interaction between a software and its user. *GUI* is the smallest category with regard to number of questions and includes question titles like *"Force GUI update from UI Thread"* and *"Object synchronization with GUI Controls"*.

[2] An entity management system automates serialization of objects (entities) for storage in database.

[3] A data scraping program downloads data from remote web URLs and stores it locally.

Table 3: Popularity measures of concurrency topics.

Topic	Avg. views	Avg. favorites	Avg. score
thread safety	2848	1.5	4.6
basic concepts	2222	1.6	4.3
task parallelism	2216	1.3	4.0
locking	2152	1.3	3.5
thread life cycle management	2130	0.7	2.4
thread scheduling	2032	0.7	2.2
process life cycle management	2004	1.0	2.6
thread pool	1841	0.9	2.7
object-oriented concurrency	1773	0.8	2.6
database management systems	1727	0.6	1.8
thread sharing	1671	0.6	2.0
GUI	1664	0.5	1.6
irreproducible behavior	1647	0.6	2.3
event-based concurrency	1636	0.7	2.3
python multiprocessing	1587	0.9	2.5
entity management	1583	0.8	2.3
memory consistency	1531	1.7	4.8
file management	1458	0.6	1.9
producer-consumer concurrency	1311	0.8	2.2
unexpected output	1304	0.5	1.6
mobile concurrency	1292	0.5	1.3
runtime speedup	1276	0.9	2.7
web concurrency	1252	0.8	1.9
concurrent collections	1155	0.5	2.0
client-server concurrency	1083	0.4	1.1
data scraping	1003	0.6	1.4
parallel computing	899	0.6	1.9
Average	*1641*	*0.8*	*2.5*

Table 4: Difficulty measurements of concurrency topics.

Topic	% w/o acc. answer	Hrs to acc. answer
database management systems	51.2	1.0
irreproducible behavior	51.1	2.1
web concurrency	50.7	0.9
mobile concurrency	50.4	0.8
client-server concurrency	50.4	0.9
python multiprocessing	50.3	0.9
parallel computation	50.1	2.1
data scraping	48.9	1.0
file management	48.8	0.6
entity management	48.0	1.8
runtime speedup	48.0	0.7
thread pool	47.0	0.7
process life cycle management	44.9	0.6
producer consumer concurrency	43.3	0.7
unexpected output	41.8	0.7
GUI	41.1	0.4
thread scheduling	40.8	0.4
thread life cycle management	40.4	0.3
locking	40.1	0.3
event-based concurrency	39.7	0.6
thread safety	39.6	0.3
concurrent collections	38.6	0.4
basic concepts	37.0	0.7
thread sharing	35.6	0.3
task parallelism	35.3	0.4
object-oriented concurrency	35.2	0.3
memory consistency	33.2	0.4
Average	*43.8*	*0.7*

Our concurrency topic *GUI* coincides with Rosen and Shihab's [28] UI topic for mobile programming.

3.3 RQ2: Popularity of Concurrency Topics

Popularity of concurrency topics is measured using average number of views of its questions, their average number of favorites and average scores, as described previously. Table 3 shows the popularity of concurrency topics using these metrics in a table, sorted by average number of views.

Intuitively, a topic with higher number of views and favorites and a higher score is more popular [22, 28, 34]. In Table 3, the *thread safety* topic has the highest views, third highest favorites and second highest score whereas *client server concurrency* has the third lowest views and lowest favorites and score.

> ***Finding 9:*** Questions about *thread safety* are among the most popular whereas *client server concurrency* questions are among the least popular.

3.4 RQ4: Difficulty of Concurrency Topics

Difficulty of concurrency topics is measured using percentage of questions with no accepted answers and average median time to get an accepted answer, as described previously. Table 4 shows difficulty measurements using these metrics in a table, sorted by percentage of questions with no accepted answers.

Intuitively, a topic with higher percentage of its questions not receiving accepted answers or taking longer to receive accepted answers is more difficult. In Table 4, the *irreproducible behavior* topic has second highest percentage of questions with no accepted answers and highest time to accepted answers whereas *memory consistency* has the lowest percentage of questions with no accepted answers and second lowest time to accepted answers.

> ***Finding 10:*** Questions about *database management systems* are among the most difficult questions whereas *memory consistency* questions are among the easiest.

3.5 RQ5: Popularity/Difficulty Correlations

As discussed, *thread safety* is among the most popular concurrency topics but its difficulty is near bottom. Intuitively, this could suggest that there may be a correlation between the difficulty and popularity of concurrency topics. We confirm this intuition using Kendall correlation by taking the following steps. First, we calculate six correlations between each of our three popularity and two difficulty metrics. Second, for each correlation, we perform a significance test at the 90% confidence level to determine if the null hypothesis, of no significant correlation, can be rejected in favor of the alternative hypothesis, that there is a negative correlation between popularity and difficulty between all three popularity metrics and two difficulty metrics. Interestingly, for all of six correlations, we find that there is sufficient evidence to conclude that its alternative hypothesis holds and there actually is a statistically significant negative correlation. Table 5 shows p-values for correlations of popularity and difficulty metrics all of which are below 0.05 except one that is below 0.1.

Unlike concurrency topics, not all Stack Overflow topics have negatively correlated difficulty and popularity. For example, Pinto

Table 5: Correlations of popularity and difficulty metrics.

p-value	*Avg. views*	*Avg. favorites*	*Avg. score*
% w/o acc. answer	0.0044	0.01166	0.001073
Hrs to acc. answer	0.001449	0.09196	0.02726

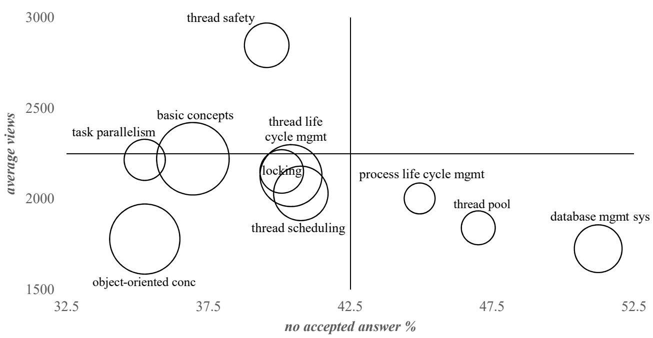

Figure 4: Trading off concurrency topics.

et al.'s [24] *Measurement* theme in their study of energy efficiency, is their most difficult and popular theme simultaneously. Similarly, the general *mobile development* topic that Barua et al. [7] finds popular is found to be difficult by Rosen and Shihab [28]. Note that the correlation between difficulty and popularity of concurrency topics does not imply causality. Wang et al. [32] study the relation of 46 factors with time to get an accepted answer to a question, which is one of our metrics for topic difficulty.

Finding 11: There is a statistically significant negative correlation between popularity and difficulty of concurrency topics.

4 IMPLICATIONS

The results of our study can not only help concurrency developers but also concurrency educators and researchers to better decide where to focus their efforts, by allowing them to trade off one concurrency topic against another based on their popularity and difficulty. To illustrate, Figure 4 shows the difficulty of our top 10 popular concurrency topics. For simplicity, popularity and difficulty equal the average number of views for questions of a topic and percentage of questions without accepted answers. In the figure, circles are topics with their size showing their number of questions.

Developers Using Figure 4, a novice concurrency developer may decide to focus their learning on *task parallelism* with higher popularity and less difficulty compared to *database management systems*. In contrast, a more knowledgeable developer who likes to learn about advanced topics with more than average difficulty may decide to learn about *process life cycle management*. Similarly, the manager of a development team can use Figure 4 to assign a less difficult task related to *task parallelism* to a more novice developer and a more difficult task related to *database management systems* to a more knowledgeable developer [34].

Educators Using Figure 4, an educator may decide to devote more material and teaching time to the more difficult *thread pool* topic compared to *process life cycle management*. Similarly, an educator can use Figure 4 to schedule teaching of the more popular and less difficult *thread safety* topic before *thread scheduling*.

Researchers Using Figure 4, a researcher may decide to focus their research project on the more difficult and slightly less popular *thread pool* rather than *thread life cycle management* in the hope of making contributions in a less crowded area.

Obviously, there are many factors that go into tradeoffs that developers, educators and researchers make to decide where to focus their efforts. However, we believe our findings can contribute to inform and improve these decision making processes.

5 THREATS TO VALIDITY

In this section, we discuss threats to the validity of our study [33].

Internal threats Use of concurrency tags to identify concurrency posts is an internal threat to validity. This is because concurrency tags may not be able to identify the complete set of concurrency-related posts. To minimize this threat we use well-known techniques used by previous work [28, 34] in developing our concurrency tags and solid experiments with a broad range of tag relevance and significance thresholds α and β. Parsing Stack Overflow dataset, inferring topics from textual contents of posts and reduction of words to their bases is another threat. To minimize this threat, we use well known tools used by previous work. We parse Stack Overflow posts using Python elementTree XML API [28], infer topics using MALLET [5, 7, 28] and reduce words using the Porter stemming algorithm [5].

External threats Use of Stack Overflow as the only dataset to study interests and difficulties of concurrency developers is an external threat. This is because Stack Overflow posts may not be a representative of developer interests and difficulties. However, Stack Overflow's large number of participant developers and posts along with its wide-spread popularity among developers may mitigate this risk. Also, we use title and body of not questions only but also their accepted answers to mitigate this risk.

Construct threats Manual labeling of topic word sets is a construct threat. To minimize this threat, we use a well-known approach used by previous work [5] to label topics using their top 10 words and 15 random questions. Determining an optimal value for K when inferring topics is another threat. To minimize this threat, we use a well-known approach used by previous work [5, 7, 28] to find a reasonable value for K using experiments with a broad range of values for K. It is well-known that determining an optimal value for K is difficult [7]. Heuristics to measure popularity and difficulty could be another threat. To minimize this threat, we use well-known heuristics and tools used by previous work to measure popularity [5, 28, 34] and difficulty [28, 31, 34].

6 RELATED WORK

Previous works that are closer to our work study software knowledge repositories, such as Stack Overflow, to understand interests [1, 2, 7, 13, 14, 28, 31] and difficulties [5, 28, 31, 34] of developers with software development topics.

Concurrency Closest to our work is the work of Pinto et al. [25] that uses the 250 most popular concurrency questions on Stack Overflow to study difficulties that developers face when writing concurrent programs. They categorize these difficulties into a set of themes including theoretical and practical concepts, threading and first steps themes, some of which coincide with our concurrency topics and categories.

In other previous work, Pinto et al. [26] analyze the code for 2227 projects to understand the usage of Java's concurrent programming constructs and libraries and the evolution of the usage. Lin and Dig [17] study a corpus of 611 widely used Android apps to understand how developers use Android constructs for asynchronous concurrency. Blom et al. [10] study the usage of java.util.concurrent library in Qualitas corpus. Godefroid and Nagappan [12] survey 684 developers to study the spread and popularity of concurrency platforms and models at Microsoft.

In contrast, in this work we develop a set of concurrency tags to extract and study a large set of 245,541 concurrency posts on Stack Overflow and use latent Dirichlet allocation (LDA) to infer concurrency topics using textual contents of these posts and organize them into a topic hierarchy. In addition, we measure popularity and difficulty of concurrency topics, study their correlations and discuss their implications.

Non-concurrency Rosen and Shihab [28] use LDA to infer mobile development topics on Stack Overflow. They study popularity and difficulty of their mobile topics and categorize developers' questions based on platforms for mobile development and type of questions that developers ask (why, what and how). Yang et al. [34] use LDA tuned with a genetic algorithm to infer security topics on Stack Overflow and manually organize their topics into five categories. They study popularity and difficulty of their security topics. Bajaj et al. [5] use LDA to infer client-side web development topics using Stack Overflow and study interests of developers in these topics and challenges they face when working with these topics. Barua et al. [7] use LDA to infer general topics on Stack Overflow. They study relations of questions and answers of these topics and evolution of developers' interests in these topics both in general and for specific technologies.

Gyöngyi et al. [13] and Adamic et al. [1] study Yahoo!Answers posts to determine developer's interests in a set of predefined categories. Hindle et al. [14] use LDA to infer topics related to development tasks from commit messages of a standalone software project and study evolution of developers' interest in these topics. Treude et al. [31] and Allamanis and Sutton [2] study Stack Overflow posts to infer types of questions that developers ask and determine their difficulties with these question types. Bajracharya and Lopes [6] study logs of Koders, a code search engine, to learn about general topic of interest in code search.

However in this work, we focus on inferring concurrency topics in Stack Overflow using LDA, organize these concurrency topics into a hierarchy, study their popularity, difficulty and their correlations and discuss their implications.

7 CONCLUSION AND FUTURE WORK

In this paper, we performed a large-scale study using the textual content of the entirety of Stack Overflow to better understand interests and difficulties of concurrency developers. We inferred topics of concurrency questions that developers ask about, organized them into a topic hierarchy and measured their popularity and difficulty. We showed how our findings not only can help concurrency developers but also education, and research and development communities that support these developers. One avenue of future work is to perform a similar study using commit logs and bug reports of publicly available concurrent software.

REFERENCES

[1] Lada A. Adamic, Jun Zhang, Eytan Bakshy, and Mark S. Ackerman. Knowledge sharing and Yahoo Answers: Everyone knows something. In *WWW '08*.
[2] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing Stack Overflow questions by topic, type, and code. In *MSR '13*.
[3] Mehdi Bagherzadeh and Hridesh Rajan. Order types: Static reasoning about message races in asynchronous message passing concurrency. In *AGERE '17*.
[4] Mehdi Bagherzadeh and Hridesh Rajan. Panini: A concurrent programming model for solving pervasive and oblivious interference. In *MODULARITY '15*.
[5] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Mining questions asked by web developers. In *MSR 2014*.
[6] Sushil Krishna Bajracharya and Cristina Videira Lopes. Analyzing and mining a code search engine usage log. *Empirical Softw. Engg. '12*, 17(4-5).
[7] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Softw. Engg. '14*, 19(3).
[8] Lauren R. Biggers, Cecylia Bocovich, Riley Capshaw, Brian P. Eddy, Letha H. Etzkorn, and Nicholas A. Kraft. Configuring latent dirichlet allocation based feature location. *Empirical Softw. Engg. '14*, 19(3).
[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.
[10] Stefan Blom, Joseph Kiniry, and Marieke Huisman. How do developers use APIs? a case study in concurrency. In *ICECCS '13*.
[11] Sally Fincher and Josh Tenenberg. Making sense of card sorting data. *Expert Systems '05*, 22(3).
[12] Patrice Godefroid and Nachiappan Nagappan. Concurrency at Microsoft: An exploratory survey. In *(EC)2 '08*.
[13] Zoltán Gyöngyi, Georgia Koutrika, Jan Pedersen, and Hector Garcia-Molina. Questioning Yahoo! Answers. In *workshop on question answering on the Web '08*.
[14] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *ICSME '09*.
[15] Hugh C. Lauer and Roger M. Needham. On the duality of operating system structures. *SIGOPS Oper. Syst. Rev. '79*, 13(2).
[16] Edward A. Lee. The problem with threads. *Computer '06*, 39(5).
[17] Yu Lin, Semih Okur, and Danny Dig. Study and refactoring of Android asynchronous programming. In *ASE '15*.
[18] Yuheng Long, Mehdi Bagherzadeh, Eric Lin, Ganesha Upadhyaya, and Hridesh Rajan. On ordering problems in message passing software. In *MODULARITY '16*.
[19] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. A comprehensive study on real world concurrency bug characteristics. In *ASPLOS '08*.
[20] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest qa site in the west. In *CHI '11*.
[21] Andrew Kachites McCallum. MALLET: A machine learning for language toolkit.
[22] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. "Jumping through hoops": Why do Java developers struggle with cryptography APIs? In *ICSE '16*.
[23] Stephan Neuhaus and Thomas Zimmermann. Security trend analysis with CVE topic models. In *ISSRE '10*.
[24] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *MSR 2014*.
[25] Gustavo Pinto, Weslley Torres, and Fernando Castor. A study on the most popular questions about concurrent programming. In *PLATEAU '15*.
[26] Gustavo Pinto, Weslley Torres, Benito Fernandes, Fernando Castor, and Roberto S.M. Barros. A large-scale study on the usage of Java's concurrent programming constructs. *J. Syst. Softw. '15*, 106(C).
[27] M. F. Porter. Readings in information retrieval. chapter An Algorithm for Suffix Stripping. Morgan Kaufmann Publishers Inc., 1997.
[28] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using Stack Overflow. *Empirical Softw. Engg. '16*, 21(3).
[29] Stack Exchange Dump. https://archive.org/details/stackexchange, June 2017.
[30] Stack Overflow. http://www.stackoverflow.com/, June 2017.
[31] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web?: Nier track. In *ICSE '11*.
[32] Shaowei Wang, Tse-Hsun Chen, and Ahmed E. Hassan. Understanding the factors for fast answers in technical qa websites: An empirical study of four stack exchange websites. In *ICSE '18*.
[33] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. 2012.
[34] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of Stack Overflow posts. *JCST '16*, 31(5).

Acknowledgements Ahmed and Bagherzadeh were supported in part by Oakland University Department of Computer Science and Research Office.