Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code

Allamanis, M., & Sutton, C. (2013). Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. (pp. 53-56). IEEE Computer Society. doi: 10.1109/MSR.2013.6624004

Why, When, and What: Analyzing Stack Overflow Questions by Topic, Type, and Code

Miltiadis Allamanis, Charles Sutton
School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK
Email: m.allamanis@ed.ac.uk, csutton@inf.ed.ac.uk

***Abstract*—Questions from Stack Overflow provide a unique opportunity to gain insight into what programming concepts are the most confusing. We present a topic modeling analysis that combines question concepts, types, and code. Using topic modeling, we are able to associate programming concepts and identifiers (like the `String` class) with particular types of questions, such as, "how to perform encoding".**

I. Introduction

Frequently software engineers and programmer hobbyists seek answers to questions using websites such as Stack Overflow. Analyzing this data [1] has the potential to provide insight into what aspects of programming and APIs are most difficult to understand. In this paper, we categorize Stack Overflow questions into two overlapping views: *programming concepts*, and *type of information* being sought.

First, using standard topic models, we categorize questions according to *concepts* such as "applets" and "games". We extend this analysis to cluster identifiers from code snippets together with text. From this analysis we make several findings, for example, that there are topics, such as memory management and compatibility issues, which do not lend themselves to the use of code snippets.

Second, we move beyond this analysis by categorizing questions by *type*. Question types represent the kind of information requested in a way that is orthogonal to any particular technology. For example, some questions are about build issues, whereas others request references for learning a particular programming language. We present a method for clustering questions by type using an unorthodox (and to our knowledge, novel) method of applying topic models removing noun phrases. so that the model focuses on the type of the question asked. A major finding is that the distribution over question types *does not vary among languages*, suggesting that we have found a domain-general categorization of questions. We can also evaluate the *orthogonality* of tools and technologies based on the *type* of problems they can solve. This can be helpful for navigating software projects alternatives.

Finally, we connect question concepts and types to perform analyses like: "What types of questions are most commonly asked about the `Date` object in Java?" In this example, the most common type is about conversion and formatting. This provides a way of analyzing the most confusing aspects of particular programming concepts.

All these findings may have implications for assistive IDE technologies, allowing for smarter context-specific question answering using external data sources such as StackOverflow.

TABLE I: Categories of Questions Used

Category	Count	Related Stack Overflow Tags
Java	281508	java, java-ee
Python	122708	python
Android	210799	android
CSS	86961	css, css3
SQL	218326	sql, mysql, sql-server, postgresql, databases
Version Control	33314	git, mercurial, svn
Build Tools	12990	ant, maven

II. Question Topic Models

In this paper we will train three topic models to find question concepts, code and text topics and types, using Latent Dirichlet Allocation (LDA) [2]. LDA is a generative model for describing documents as mixtures of topics, with each topic containing frequently co-occurring words. To train LDA, we used MALLET [3]. For natural language processing tasks, such as text tokenization, part-of-speech tagging and chunking we used Apache OpenNLP. We stem all natural language tokens with the Snowball Stemmer [4]. Finally, for extracting Java code tokens, such as identifiers, we use Eclipse JDT. From the Stack Overflow dataset we used the questions containing a selected set of tags listed in Table I. The questions include a variety of programming languages and software engineering tools. Our goal is to find differences among categories and identify *concepts and types* of questions.

A. Question Concepts

Now, using a topic model we will identify different question concepts. We train a topic model for 2000 iterations with 150 topics on all the questions and their respective answers. This number of topics allows sufficiently granular topics to be discovered. Table II shows a sample of the topics discovered. The concepts are an indication about *what is confusing* but do not show what users are trying to achieve. We observe that LDA has found interesting clusters (word co-occurrences). For example, there are general topics with words such as "try" or "wrong" (A1), but there are problem-specific topics (e.g. databases in A4) or references to specific technologies (e.g. Java in A8). However, it is not clear from these topics *why* the users are confused; this is addressed in the next section.

Figure 1 shows a heatmap of the average topic assignment for the category questions (Table I), normalized by the maximum average topic assignment. We observe that the Java topic

TABLE II: Sample Question Concepts

	Top Words
A1	code, work, try, problem, change, correct, wrong
A2	http, link, example, tutorial, read, check
A3	code, write, make, example, easier
A4	query, table, row, join, select
A5	perform, time, cache, faster, slow, optimize
A6	page, load, javascript, iframe, html
A7	android, app, mobile, device, iphone
A8	java, application, applet, jdk, jvm
A9	return, result, function, call, method
A10	image, background, color, picture, image
A11	server, client, send, socket, data, connect
A12	size, screen, width, height, resize
A13	report, print, pdf, word, document, generate
A14	language, locale, translate, english, barcode

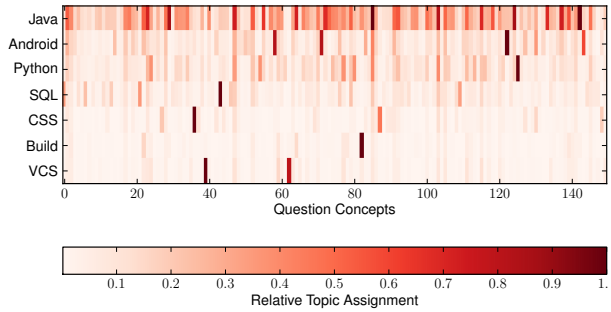

Fig. 1: Average topic assignment for question concepts normalized by the maximum average assignment. Most frequent topics are on the left.

distribution is spread out compared to the other categories. We also observe that programming languages (Java, Python) have a wider mixture of concepts. On the other hand, specialized languages, such as SQL and CSS are confined to a smaller range of topics, *mostly disjoint* from those used in Java and Python questions. Finally, build and version control systems (VCS) are dominated by few topics—uncommon with the rest of the tools.

Results: These are interesting observations indicating that we can create tools based on topic models that can *evaluate the orthogonality* of different languages, platforms and tools. Two technologies are orthogonal if they are used to solve different classes of problems. For example, the cosine similarity of the average topic assignments between Python and Java is 0.7, Java and Android is 0.61, while Java and VCSs have a cosine similarity of 0.24 being—unsurprisingly—"more" orthogonal. The average topic assignments also provide insights about the *categories of programming tasks that are more common in various languages*: For example, "log", "console", "file", "encrypt" are more common in Java than in Python, whereas "script", "shell" are more common in Python.

B. Code & Text Model

In addition to text, many questions contain code snippets. In this section, we incorporate code identifiers into the topic model. First, by tokenizing the title and—whenever possible—parsing the Java code snippets we identify that 28.2% of the question titles contain type identifiers, 16.2% variable identifiers and 7.9% method identifiers. It seems that although type identifiers are the least common in code corpora [5] *they are the most frequently asked code artifacts*. The percentage of variable identifiers is unexpectedly higher than that of methods and this can be attributed to the semantics variable names have, allowing for better code understanding.

TABLE III: Sample Code and Text Java Topics of Questions

	Top Words*
C1	way, want, work, look, something, solution, create, make, question
C2	java, library, implement, api, look, want, code, way, standard
C3	method, call, parameter, pass, function, class, invoke, return
C4	`system`, `out`, `println`, `string`, `main`, `java`, `exception`
C5	`get`, `set`, `to`, `add`, `override`, `instance`, `equals`, `new`
C6	`list`, `array`, list, `add`, `size`, `util`, `linked`, `vector`
C7	version, java, jdk, release, new, support, old, compatibility, latest
C8	`stream`, `input`, `output`, `io`, `write`, `read`, `byte`
C9	thread, `thread`, run, call, `runnable`, execute, wait, start
C10	test, `test`, unit, junit, run, mock, `assert`, `mock`, class, fail
C11	string, `pattern`, match, regex, `matcher`, `string`, `replace`
C12	`android`, `activity`, `intent`, `app`, `manager`, `context`
C13	memory, heap, jvm, size, garbage, run, gc, allocate, space, profile
C14	`audio`, play, `media`, video, audio, sound, `player`, `sound`
C15	algorithm, search, index, graph, find, implement, lucene

* Words in `typewriter font` are code identifiers.

We train a new LDA model with 150 topics (2000 iterations) but include both words and code identifiers, after splitting identifiers into parts based on underscores and capital letters. The topics discovered are shown in Table III). Surprisingly, we find topics where code identifiers prevail, others where text prevails and others where code identifiers and text co-exist. Again, some topics are general and represent the basic vocabulary for expressing questions and solutions (C1). Topics (e.g. C5) containing common Java identifiers are also present. Additionally, the majority of identifiers are type and method names but we also find commonly used variable names (e.g. `tmp`, `i`).

Results: This reaffirms that Stack Overflow questions are *about the code and are not application domain specific*, since we find that *type and method identifiers are conceptually consistent, while variable names are not*. Interestingly, we also find task-specific identifiers (C10, C14). Finally, we recognize question concepts that cannot be explained with code since they do not consistently coocur with identifiers. These are *emergent code properties*, such as memory or version issues (C7, C13).

C. Question Types

Previously, we identified concepts across questions and code that provided indications about *the task* that the users tried to achieve but not the cause of the problem. To address this issue, we focus on the context to find different *types of questions* that are not specific to any technology. By question types we mean the set of reasons questions are asked and what the users are trying to accomplish.

To build the question type model, we preprocess the text in a special way. We chunk the question and answer text and remove chunks that represent noun phrases and consider all other chunks, such as verb phrases, as single tokens. For example, "try to insert" is a single token. Verbs are

TABLE IV: Question Type Topics Sample

	Top Chunks Phrases
B1	to use, can use, to do, want to use, to get, can do, instead of
B2	doesn't work, work, try, didn't, won't, isn't, wrong
B3	run, happen, cause, occur, fail, work, check, to see, fine, due
B4	create, to create, is creating, call, can create, add, want to create
B5	hope, make, understand, give, to make, work, read, explain, check
B6	faster, run, will be, slow, depend, make, can be, fast, would be
B7	return, pass, call, to pass, is returning, is passing, will return
B8	change, update, to change, modify, can change, want to change
B9	join, select, base, return, to get, to select, filter, query, match
B10	calculate, find, give, to calculate, will be, equal, to find, compute
B11	run, connect, to connect, is running, start, configure, work, fail
B12	convert, to convert, format, encode, back, need to convert, decode
B13	build, compile, include, link, to build, to compile, run, make
B14	log, redirect, to redirect, is logged, check, to login, login, visit
B15	generate, to generate, create, is generated, produce, automatically
B16	learn, to learn, start, read, understand, recommend, find, good
B17	deploy, run, publish, build, to deploy, install, develop, host, app
B18	merge, commit, push, back, git, create, check, make, clone
B19	import, export, to import, to export, create, format, to excel
B20	sort, to sort, order, want to sort, base, sorted, compare

stemmed. Now, we have a set of phrases, mostly verb phrases, representing activities that users are trying to address. We note that since chunking and part-of-speech tagging are automatic, they will make a few errors. However, this is a minor issue. Applying topic models in this way is unorthodox and novel, since usually the emphasis is given on nouns and single verbs.

We train LDA with 100 topics (2000 iterations) using *all* Stack Overflow questions irrespective of their tag. We chose 100 topics since we need less granular question types to be discovered. Excitingly, we now have a much more informative clustering of the questions (Table IV), finding some major question categories about:

- concepts that have been coded but *do not work* (B2,B3);
- not understanding *how/why something works* (B5);
- not knowing how to *implement something* (B4);
- the *way of using* some piece of code or API (B1);
- suggestions for *learning* a language or technology (B16).

We also find general, non-technology specific issues that evoke questions such as execution speed (B6), compiling (B13), user authentication (B14), importing and exporting data (B19). Furthermore, common operations such as sorting (B20) and calling functions (B7) are also included. Thus, it is possible *to cluster questions by type rather than content*. These topics provide valuable information about problems practitioners face. The aforementioned issues seem to be calling for more organized learning resources or more descriptive, intent-specific APIs or better documentation searchability.

Figure 2 shows the assignment distribution of types across the categories. Compared to Figure 1, among Java, Android and Python we now get similar types of questions. Specifically, Java and Python are very similar (cosine similarity 0.98) and Android differs only slightly (cosine similarity with Java 0.95 and 0.92 with Python). Thus, the types of questions practitioners have in programming languages are similar and independent of the language, at least between Python and Java.

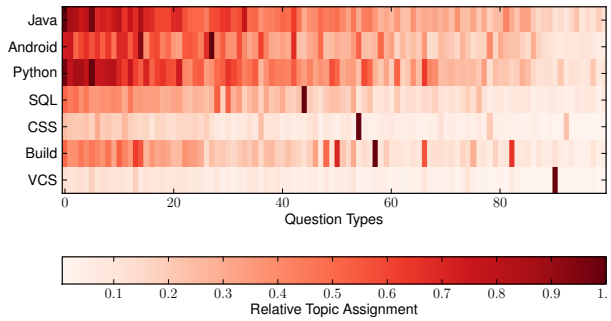

Fig. 2: Average topic assignment for question types, normalized by the maximum average assignment. Most frequent topics are on the left.

Platforms (like Android) slightly change the distribution *refocusing* it on a specific domain (mobile devices for Android).

For most of the technologies studied in this paper, generic phrases like "to use" or "can do" rank high with minor differences. However, interesting differences also arise: for example questions containing phrases such as "implement", "create", "handle", "deploy" and "override" are more common in Java, compared to Python. Similar observations are made with question types having phrases such as "concurrently", "synchronize" and "serialize". On the other hand question using verbs such as "calculate", "compute", "parse", "match", "replace", "extract" and "import" are more common in Python compared to Java. We hypothesize that this happens because Python is commonly used for such tasks. Similar results are also observed between Android and Java questions. Unsurprisingly, Android questions are much more about displaying, rendering, loading and clicking compared to Java.

We also find that *task-specific technology questions are focused*: Build technologies are used with words like "build", "link" and "deploy" (B13) and VCSs with words like "merge" and "commit" (B18). SQL questions focus on "filtering" and "storing" data (B9), but speed (B6) seems to be an import issue. Conversely, CSS is mostly concerned with verbs about placing items on a screen. The cosine similarity of question types between CSS and VCSs is 0.41 indicating widely different types of questions. Interestingly, CSS seems to be easily tweaked since the topic containing words such as "work", "change", "does not work", "wrong" is ranked higher than the other languages. According to Figure 2, SQL and build tools are more similar to programming languages compared to VCSs and CSS. **Results:** From the discussion above it seems that the question type model, could be helpful for code-related information retrieval allowing better coding question understanding and code-aware query expansions.

III. Connecting Question Concepts and Types

Now, we connect the topic models of Table III and Table IV by examining how different topics were assigned to the same documents. We study the covariance of the topic assignments to *identify what are the confusing types and topics of questions*. Looking at relatively large differences in the covariances of the question concepts and types among the categories, we

TABLE VI: Percent (%) of questions asked on a specific day for various tags.

	Java	Java EE	Android	JDBC	Python	C#	RoR	SQL Server	C++	Maven	.NET	iPhone	XML	**All**
Mon	15.9	16.5	16.1	15.5	15.2	16.0	15.5	16.7	15.3	15.9	16.0	16.1	16.0	**15.6**
Tue	17.3	18.0	17.1	18.8	16.7	18.0	17.0	19.0	16.5	18.8	18.0	17.5	18.0	**17.4**
Wed	17.5	18.6	17.2	17.5	17.0	18.1	16.8	19.5	16.6	17.8	18.6	17.4	18.3	**17.6**
Thu	17.2	17.2	17.0	18.0	16.5	17.9	16.5	19.3	16.7	18.8	18.2	16.9	18.0	**17.4**
Fri	15.3	15.3	15.2	14.8	14.9	15.7	15.0	16.3	15.0	16.1	16.0	15.3	15.4	**15.7**
Sat	8.3	7.5	9.0	7.5	9.7	7.2	9.5	3.6	9.8	6.3	6.6	8.8	7.0	**8.3**
Sun	8.5	7.0	8.3	7.9	9.9	7.1	9.8	5.7	10.1	6.4	6.6	8.1	7.2	**8.1**

TABLE V: Strong Correlations between Text & Code Topic with Question Type

Topic*	Question Type Topics
character, encoding, string, ascii	convert, encode, format, decode
`date`, `format`, `time`, `parse`	convert, encode, format, decode
thread, `Runnable`, call, task	start, run, wait, call, kill
time, performance, memory, large	faster, run, slow, depend
class, method, interface	implement, inherit, call, create
`key`, `ssl`, certificate, `cipher`	hash, encrypt, store, sign
`audio`, play, `media`, video, `sound`	load, play, reload, download
`color`, `width`, `height`, `paint`	draw, rotate, render, transform

* Words in `typewriter font` are code identifiers.

find that questions containing problems with "crashes" in the "browser", an "applet" or a "web service" are uncommon in Python. Compared to Java, Android questions contain less frequently tokens such as "results", "search", "store" than phrases such as "submit", "enter", "upload". This is happening since search servers are rarely implemented in Android, while search facilities are much more common in Java applications.

The observations above are interesting for software engineering and specifically to software design. *Joint topic models can be part of tools that help with technology selection based on the "actions" imposed by the requirements*. Additionally, they can help discover similar technologies (e.g. build systems) as an initial "market research" *widening the pool of potential tool alternatives considered*. Interestingly, the combination of these topic models can be helpful to code search.

We now focus on the covariance between the Code & Text concepts (Table III) with the question types (Table IV) and look for topics that co-occur frequently (higher absolute covariance). The covariance seems to be uniform across topics, with some exceptions: For example, we find higher covariance between question types containing words such as "start", "run", "stop" with the topic containing `Runnable`, "thread", `run`, "call" etc.; the topic containing "string", `pattern`, "regex" is correlated with phrases such as "match", "replace" and "escape". Interestingly, we also find correlations between words such as "code", "pattern", "implement", "design" associated with the phrases "make", "easier", "write", "find" indicating software design questions.

Results: From the top correlations we are able to determine what are the most confusing or problematic issues about specific identifiers and concepts as shown in Table V. This is an interesting observation indicating that we *can predict the type of question asked from few keywords about the domain, a useful feature for IDEs and information retrieval systems*.

IV. Hobby or Serious Work?

As a side issue, we now examine programming language usage on a daily basis. Table VI shows the percentages of posts for each weekday throughout the whole dataset for some tags. First, weekends are less "busy" days for Stack Overflow. We also observe that Mondays and Fridays are "slower" days.

However, it is interesting to notice differences across languages and technologies during the weekend. Those that are used more in corporate environments (such as SQL Server, Java EE and .NET) have a much lower percentage of questions asked during the weekend. On the other end, we find that C++, Python and Ruby on Rails (RoR) have a larger percentage of questions during weekends. We hypothesize that this effect is attributed to the technologies more *widely used by hobbyists*[1], since such technologies include popular scripting languages and low entry-barrier technologies (e.g. Android).

V. Conclusion

In this paper, we observed how topic models provide intuitions about programming languages and the problems practitioners face. The question type analysis allowed us to make findings that would not have been possible otherwise. Most notably, we were able to show that the types of questions asked do not vary across programming languages, and we presented a method for identifying what question types were mostly associated with particular programmming constructs/ identifiers. In the future, IDEs and smart documentation systems using this information and the identifiers in the current context could be able to provide guidance and context-specific answers to frequently asked questions.

Acknowledgment

The authors would like to thank Prof. Andrew D. Gordon for his insightful comments and the anonymous reviewers for their suggestions. This work was supported by Microsoft Research through its PhD Scholarship Programme.

References

[1] A. Bacchelli, "Mining Challenge 2013: Stack Overflow," in *The 10th Working Conference on Mining Software Repositories*, 2013, p. to appear.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[3] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002.

[4] M. F. Porter, "Snowball: A language for stemming algorithms," 2001.

[5] M. Allamanis and C. Sutton, "Mining Source Code Repositories at Massive Scale using Language Modeling," in *The 10th Working Conference on Mining Software Repositories*, 2013, p. to appear.

[1] They could also be workaholics and academics but it is impossible to tell.