# StackInTheFlow: Behavior-Driven Recommendation System for Stack Overflow Posts

Chase Greco
Virginia Commonwealth University
Richmond, Virginia, USA
grecocd@vcu.edu

Tyler Haden
Virginia Commonwealth University
Richmond, Virginia, USA
hadentj@vcu.edu

Kostadin Damevski
Virginia Commonwealth University
Richmond, Virginia, USA
kdamevski@vcu.edu

## ABSTRACT

Developer behavior in the IDE, including commands and events and complementing the active source code, provides useful context to in-IDE recommendation systems. This paper presents StackInTheFlow, a tool that generates interpretable queries to Stack Overflow, and recommends Stack Overflow posts when a developer is observed to be facing difficulty, defined by encountering error messages or not appearing to make progress. StackInTheFlow monitors clicks on its retrieved results, and, over time, personalizes the retrieved posts to a specific set of Stack Overflow tags.
**Video:** `http://bit.ly/stackintheflowdemo`

## CCS CONCEPTS

• **Software and its engineering → Integrated and visual development environments**; • **Information systems** → *Personalization*; *Social recommendation*;

## KEYWORDS

Stack Overflow, IDE, recommendation system

**ACM Reference Format:**
Chase Greco, Tyler Haden, and Kostadin Damevski. 2018. StackInTheFlow: Behavior-Driven Recommendation System for Stack Overflow Posts. In *Proceedings of International Conference on Software Engineering (ICSE'18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

Software developers frequently search for Web resources in order to learn from others, and sometimes even to remind themselves of details related to development knowledge they already possess [1]. The Stack Overflow Q&A forum, and its large archive of software-related posts, has continued to grow in popularity with software developers, with over 40 million monthly visitors, including an estimated 16.8 million professional developers and university students [10].

Recommendation systems can aid developers in managing the large information requirements of modern software development [9]. Several recommendation tools targeting Stack Overflow have been proposed with the aim of improving developer productivity by integrating relevant information from Stack Overflow into the IDE. Prompter [8] and Seahawk [7] are able to automatically recommend Stack Overflow posts based on source code context present in the IDE. T2API [6] and NLP2Code [2] recommend code snippets extracted or adapted from Stack Overflow based on natural language text describing the programming task.

Opportunities still exist for such tools to better integrate with the IDE and to the developer's behavior, further personalizing and targeting recommendations to opportune moments in time. In this paper, we introduce StackInTheFlow- a tool that intends to automate the manual task of finding relevant Stack Overflow posts. StackInTheFlow is personalized to each developer and integrates closely with their IDE behavior, allowing developers to remain in a high-productivity *flow* [5]. StackInTheFlow has the following set of characteristics: 1) automatically constructs interpretable queries based on the current source code context; 2) uses clicks on retrieved results to personalize, over time, the retrieved Stack Overflow posts to specific Stack Overflow tags; 3) automatically recommends Stack Overflow posts on compiler and runtime errors in the IDE; 4) detects when a developer is facing difficulty and not making progress and recommends Stack Overflow posts. 5) queries the Stack Overflow API (and not the periodic dump) to retrieve the most recent Stack Overflow posts.

StackInTheFlow integrates as a plugin with the IntelliJ family of Java IDEs, including the popular Android Studio environment. Though the tool targets a Java IDE, the mechanisms it uses are language agnostic and can be generalized to other languages with minimal effort. In this paper, we describe each of StackInTheFlow's features, and include a set of preliminary results on the effectiveness of each recommendation mechanism using field data gathered from use of the tool by developers.

## 2 TOOL USE CASES

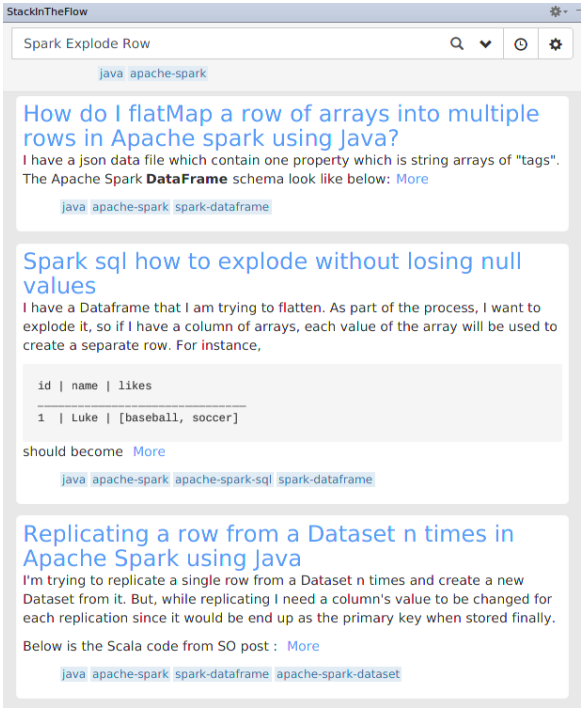
StackInTheFlow use cases can be organized by the type of query that initiates them: manual, automatic, error and difficulty. To illustrate StackInTheFlow's use, consider a scenario where a developer has just begun using the Apache Spark parallel programming framework.

**Manual Query.** Utilizing StackInTheFlow, the developer may manually write and issue queries to the Stack Overflow API, such as regarding the Spark *explode* operation (as in Figure 1). From there she may browse the results of her search directly within the IDE, adding tags to filter the results, or sort them based on four different criteria: relevance, newest, last active, and number of votes, mirroring the functionality of the Stack Overflow website. If a particular result seems relevant to the problem at hand, she may expand it to view the full question by clicking *More* or view the full post within the browser by clicking the title of a result. Finally, she


*ICSE'18, May 2018, Gothenburg, Sweden*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

**Figure 1:** StackInTheFlow **User Interface.**




may view and revisit her previous searches by selecting the *History* tab.

**Automatic Query.** Occasionally, the developer may not be able to form a query suitable to retrieve the information required to solve a development problem and may require assistance in query composition. For example, the developer may wish to know how to set the configuration options for the *SparkSession* object. In such a case she may simply highlight the section of code relevant to declaring or using this object, right-click and select the *Auto Query* option. Utilizing the procedure detailed in Section 3.1, StackInTheFlow will automatically generate a query from the code snippet and present the results in the same fashion as above.

**Error Query.** Inevitably, during the course of their daily work a developer will encounter error messages. These messages can often be cryptic and unfamiliar to the developer, requiring the consultation of sources such as Stack Overflow in order to decipher their meaning. To address this issue, whenever an error message is encountered, either during compile or run time, StackInTheFlow will generate a query and recommend results using the approach described in Section 3.2.

**Difficulty Query** Finally, it may be the case that the developer is stuck in an unproductive loop. She may be deleting large portions of code without making significant progress, or scrolling through files without making any edits. StackInTheFlow contains mechanisms to detect such behaviors, outlined in Section 3.2, and to automatically generate queries and present recommendations without any direct input from the developer, which may provide the information they need to overcome their development block.

# 3 TOOL DESCRIPTION

The different components of StackInTheFlow and their relationship to the different use case queries is shown in Figure 2. Here, we describe the internals of each of these key components of the tool.

## 3.1 Query Generation

StackInTheFlow reduces the burden on the developer of composing queries to Stack Overflow by providing facilities for the extraction of query terms from the source code within the active editor in the IDE. When the automatic query generation functionality is invoked, either manually by the user or automatically as shown in Figure 2, StackInTheFlow generates and issues a user interpretable query based on (a selection of) the currently active source code. To generate the query, StackInTheFlow matches terms extracted from the source code with a stored dictionary of terms pre-mined from the user posts contained within the periodic Stack Overflow Data Dump. Since the dictionary is computed offline, query generation is lightweight and fast.

If the user has selected a subsection of the source code when invoking the StackInTheFlow auto query command, terms are extracted from it, otherwise terms are extracted from the `import` statements of the document, if available, as well as the lines surronding where the cursor resides. Each package level of the `import` statement is treated as a separate term.

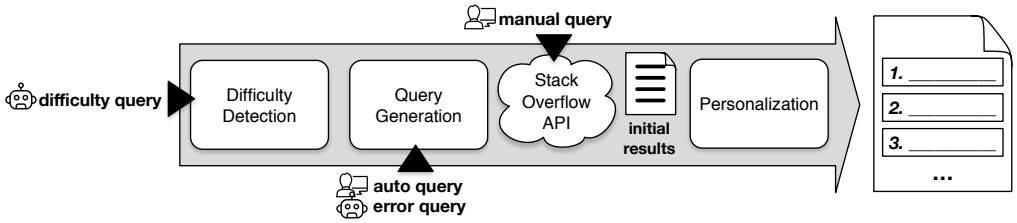
StackInTheFlow's dictionary contains a set of query pre-retrieval metrics [3], which enable the selection of the terms that are most likely to retrieve a reasonable selection of documents from Stack Overflow. Several key characteristics of query terms, *specificity*, *similarity* and *coherence*, are used to select the most effective set of query terms.

The *specificity* of a query is a measure of the distribution of the terms within the corpus. For queries composed of generic terms which occur frequently within the corpus, for example "the" and "there", the quality is deemed lower than that of queries containing specific terms such as "apache" or "spark". To assess the specificity of the candidate terms we calculate both the Inverse Document Frequency and the Inverse Collection Term Frequency for each term.

The *similarity* of a query is a measure of the similarity between its terms and the corpus, with the intuition that queries judged similar to the corpus are easier to answer and thus of higher quality. For each of our candidate terms, we rely on the *Collection Query Similarity* metric to ascertain a similarity score.

The *coherence* of a query is a measure of the inter-similarity of documents containing the query terms. Several metrics have been proposed to examine this dimension, however most are computationally expensive to compute as they require a pointwise similarity matrix for all documents contained within the corpus. We instead employ a less expensive metric *VAR(t)*, which measures the variance of the query term weights over the documents which contain them. To compute the weight of each query term we utilize a *tf-idf* based approach. The argument behind such metrics being that if the variance is low, it will be more difficult to differentiate between relevant and irrelevant documents, thus the overall query quality is low.

**Figure 2: Overview of StackInTheFlow.**

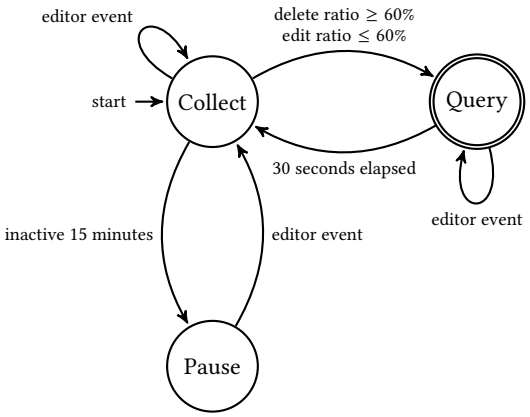



For each candidate source code term contained within the dictionary, we calculate the above metrics treating each term individually as a query, and then linearly sum the score for each dimension to achieve an overall score. The top four scoring terms are selected to form the candidate query which is then used to query Stack Overflow. If no results are returned a back-off technique is employed by incrementally removing terms from the query.

## 3.2 Recommendation System

StackInTheFlow automatically suggests Stack Overflow posts to the developer when she encounters obstacles to completing her task. One such case is when compiler or runtime error messages are encountered. While it is common for parts of compiler error messages and runtime stack traces to be included in Stack Overflow questions, error messages are not standardized for searching related documentation. StackInTheFlow translates the contents of an in-IDE error message into a query utilizing two separate mechanisms. The first mechanism extracts the first line of the stack trace as the raw query. If this does not retrieve any Stack Overflow posts, a more complex approach is employed, combining terms extracted with regular expressions for determining exception classes and language version, with the stack trace, which is tokenized and filtered by the term dictionary described in Section 3.1. The top four resulting terms are used to query Stack Overflow. As before, we use a back-off technique of incrementally removing less-salient terms when no results are retrieved.

**Figure 3: Difficulty Detection State Machine.**




In the case of an error message, the indication of the user encountering an obstacle is quite clear, however, the user may encounter difficulty without as clear an indicator. They may be constantly editing the same section of code, or unable to continue development due to lack of project understanding. StackInTheFlow provides mechanisms to detect such cases and to provide Stack Overflow posts to aid the developer, by leveraging its automatic query generation capability. It does this by taking an approach similar to that of Carter and Dewan [4], by analyzing the ratios between the insertion and deletion of text within the editor. Such an approach, Carter and Dewan argue, prevents the tool from falling into the trap of determining the programmer has encountered difficulty when they have simply *"gotten up to get a cup of coffee"*. This approach utilizes the finite state machine detailed in Figure 3.

The machine begins in the *Collect* state. While in this state, editor events are collected. Editor events are broken into four categories: *Insert*, *Delete*, *Scroll*, and *Click*. *Insert* and *Delete* events are fired when the user inserts or deletes a character respectively. *Scroll* events are fired when the user scrolls the view within the editor. *Click* events are fired when the user clicks the mouse. A queue of the past 25 events is maintained. In order to control for event bursts, consecutive events of the same type within one second are ignored and not added to the queue except for the initial event. If at any time the ratio of *Delete* events to *Insert* events crosses 60%, the machine determines that the programmer has encountered difficulty, and a query is generated, returning a set of Stack Overflow results to the developer, while the machine transitions to the *Query* state. In addition to the insert/delete ratio proposed by Carter and Dewan [4], we utilize a second novel threshold, based on the ratio between the number of edit events (insert, delete) and non-edit events (scroll, click). If this ratio falls below 60%, the machine will also determine the programmer has encountered difficulty, suggest results, and transition to the query state. The rationale behind this second threshold is that if the user is spending the majority of their time moving the cursor without editing text, they have likely encountered a difficulty.

While in the *Query* state, the event queue is cleared and all subsequent editor events are ignored until 30 seconds has elapsed, at which point the machine transitions back to the *Collect* state. Finally, if the user has been inactive for at least 15 minutes, i.e. no editor events are being generated, the machine transitions to a *Pause* state, where it remains until an editor event occurs.

### 3.3 Result Personalization

StackInTheFlow personalizes the results of all queries by re-ranking the results based on past user activity. It does this by utilizing a novel metric, *Click Frequency-Inverse Document Frequency* (*cf-idf*), which aims to predict the affinity of a developer towards a retrieved Stack Overflow post by analyzing the tags associated with previously clicked results by that developer. *Click Frequency-Inverse Document Frequency* is composed of two constituent metrics. The first, *Click Frequency*, cf($t$,$H$), is computed for a given tag $t$ on a given result selection history $H$. Using raw frequency $f_{t,H}$, defined as the number of times a Stack Overflow post tagged with $t$ was clicked on by the user, as recorded in $H$, the corresponding *Click Frequency* is given by the following equation.

$$\mathrm{cf}(t,H) = \begin{cases} 1 + \log(f_{t,H}) & f_{t,H} > 0 \\ 0 & f_{t,H} = 0 \end{cases}$$

The second metric is the *Inverse Document Frequency* of a tag, computed from the Stack Overflow Data Dump, which discounts tags that are prevalent in the corpus. From these two metrics we calculate *Click Frequency-Inverse Document Frequency* (*cf-idf*) in the same way one would calculate *tf-idf* by computing their product.

Given an initial set of of ranked results retrieved from Stack Overflow, for each result we compute a raw score by taking the sum of the *cf-idf* of each tag associated with the result and dividing by the total number of tags associated with the result. Finally, we sort the Stack Overflow results based on an average of the rank of each result in the original retrieval ranking with its corresponding weighted rank.

## 4 PRELIMINARY RESULTS

At the time of submission, StackInTheFlow, with all the features described in this paper, has been publicly available for download from the JetBrains tool repository for 3 months. Following advertising of the tool on various channels, it has been downloaded 148 times, with logs reflecting tool use captured from 52 unique users. Logs from StackInTheFlow developers have been marked and removed.

The goal in our preliminary evaluation is to estimate the effectiveness of each query type. To perform this evaluation we rely on the feature usage and click-through rates found in the logs. Our assumption is that clicking on a query's result to open it in the browser or expand its content for reading within StackInTheFlow is an indication of its effectiveness.

Out of the 635 queries we logged, 55% came from manual user input, 17% from difficulty detection, 16% from user action events, and 12% resulted from automatic error message queries. We consider the manual query as the baseline query type, and compare the other query types against this one.

Our logs contain 153 instances of user-result interactions. Out of these, users opened the browser 74/153 (48%) times , and either expanded or contracted articles within the tool 79/153 (52%) times. In correlating the StackInTheFlow user-result interactions with a query type, we observe that 125/153 (81%) are on manual queries, 4/153 (3%) are on automatically generated queries, 9/153 (6%) are on difficulty queries, while runtime error queries account for 14/153 (9%) of user interactions.

Finally, we explore how often users interact with retrieved Stack Overflow posts based on query generation type. Out of 349 manual queries, 48 (14%) had at least one click. From 103 automatically generated queries, 1 (1%) had at least one click. Out of 105 difficulty queries, 8 (8%) had at least one click. Out of 78 error queries, 9 (12%) had at least one click.

In summary, our initial results indicate that automatic queries rarely received user interaction. However, a common user case is for automatic queries to serve as a starting query, which is later reformulated manually by the user. Both error (12%) and difficulty (8%) queries resulted in at least a single click with a frequency on par with manual queries (14%), which indicates that they were reasonably effective.

## 5 CONCLUSIONS

This paper introduces StackInTheFlow, a tool that allows for in-IDE querying of Stack Overflow. StackInTheFlow also recommends Stack Overflow posts to developers using several mechanisms, including on the occurence of compilation or runtime errors and when the tool detects the programmer is having difficulty, based on key presses and clicks in the IDE. In our preliminary evaluation, we observe that the StackInTheFlow's recommendations are effective, receiving clicks on retrieved Stack Overflow posts generally as often as manually composed queries.

## ACKNOWLEDGEMENTS

We would like to thank John Coogle, Jeet Gajjar, and Kevin Ngo for contributing to the implementation of StackInTheFlow and the support of the DARPA MUSE program under Air Force Research Lab contract FA8750-16-2-0288.

## REFERENCES

[1] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. 1589–1598.

[2] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. In *Proceedings of the 2017 International Conference on Software Maintenance and Evolution*.

[3] David Carmel and Elad Yom-Tov. 2010. *Estimating the Query Difficulty for Information Retrieval*. Morgan & Claypool.

[4] Jason Carter and Prasun Dewan. 2010. Design, Implementation, and Evaluation of an Approach for Determining when Programmers Are Having Difficulty. In *Proceedings of the 16th ACM International Conference on Supporting Group Work (GROUP '10)*. 215–224.

[5] C. S. Corley, F. Lois, and S. Quezada. 2015. Web Usage Patterns of Developers. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 381–390.

[6] Thanh Nguyen, Peter C. Rigby, Anh Tuan Nguyen, Mark Karanfil, and Tien N. Nguyen. 2016. T2API: Synthesizing API Code Usage Templates from English Texts with Statistical Translation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. 1013–1017.

[7] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack Overflow in the IDE. In *Proceedings of the 2013 International Conference on Software Engineering*. 1295–1298.

[8] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 102–111.

[9] Martin P Robillard, Walid Maalej, Robert J Walker, and Thomas Zimmermann. 2014. *Recommendation Systems in Software Sngineering*. Springer Science & Business.

[10] Stack Overflow Developer Survey. 2017. https://insights.stackoverflow.com/survey/2017. (2017).