Do developers search for source code examples using multiple facts?

Reid Holmes
Department of Computer Science & Engineering
University of Washington
Seattle, WA, USA
rtholmes@cs.washington.edu

Abstract

In this paper we examine the search behaviours of developers using the Strathcona source code example recommendation system over the period of three years. In particular, we investigate the number of query facts software engineers included in their queries as they searched for source code examples. We found that in practice developers predominantly searched with multiple search facts and tended to constrain their queries by iteratively adding more facts as needed. Our experience with this data suggest that example search tools should both support searching with multiple facts as well and facilitate the construction of multi-fact queries.

1. Introduction

Several research tools have been created to help developers locate relevant source code examples [7, 4, 1, 3, 6, 5]. One dimension in which these systems vary (see Section 2 for additional detail) is whether they allow a single search fact (such as `getStatusLine()`) or multiple search facts. Although the precision of searches can be improved by providing multiple search facts, the question of whether developers know enough information to create multiple search facts, and whether they are willing to enter them, remains open. By analyzing log files from hundreds of query sessions by nearly one hundred users of Strathcona, a system that allows developers to query for source code examples using multiple search facts, we present preliminary evidence that developers have the knowledge to formulate queries with several search facts in their search for source code examples and do so in practice.

Strathcona is a client–server example recommendation system that enables developers to quickly select a block of code from which a query is automatically generated. The server then returns source code examples that best match the code the developer has selected [1, 2]. We are able to analyze Strathcona usage patterns as the server component saves to disk every query made by developers as well as their corresponding responses.

The primary contribution of this paper is evidence that developers searching for source code examples usually provide multiple search facts in practice. We have observed that 92% of queries contain two or more facts while 36% of queries contain five or more facts. Investigating individual query sessions we found that developers queried on average 2.5 times per query session and often augmented their previous queries with new facts learned from prior results.

Background details on related search approaches is given in Section 2. Section 3 provides a brief overview of the Strathcona tool. The data we analyzed and some quantitative results are presented in Section 4. The paper ends with some suggestions for future code search tools (Section 5) and conclusion (Section 6).

2. Related Work

Several research tools have been developed that can help developers locate relevant source code examples. CodeBroker is an adaptive system that automatically queries an example repository using the comment and method signature of the method the developer's cursor is currently in [7]. Prospector locates examples given a start and end types; the tool then computes possible paths that would enable a developer to get a reference to the end type given their starting type by statically mining example source code [3]; PARSEWeb [5] uses the same input and locates examples using existing code search engines. CodeWeb [4] and MAPO [6] take a simple input and locate examples using generalized association rules.

Each of these systems constrains the number of facts that can be queried on by the developer; typically at most two query facts can be specified although often one of these is reserved. CodeBroker uses one fact for the method signature and the other for the method comment; these cannot be changed by the developer except by moving the cursor to another method. Prospector and PARSEWeb specify that one fact is related to the origin of the query and the other as

the destination. Both CodeWeb and MAPO generate relevant examples from a single query fact.

In contrast to these approaches, Strathcona allows the developer to select any contiguous block of code; all of the statically derivable facts that can be extracted from this block are automatically collected sent to the server in the query. The developer can adapt the query by modifying their selection, but cannot modify the the query otherwise.

3. Strathcona

The Strathcona example recommendation system is an Eclipse plug-in that helps developers search for source code examples. Strathcona is unique in its mechanism for automatically constructing queries for the developer based on their development context. The extracted facts are sent to a remote server that contains a repository of source code; using a series of heuristics [2] the server identifies examples that best match the developer's query.

Strathcona returns at most 10 matches, regardless of the number of examples that are located. The developer can view an abstract representation of each example using a UML-like view, requesting to see the source code only if the example seems relevant to their task. As Strathcona queries are constructed automatically, we envisioned that developers using the tool would query on many structural facts. While we have shown that the heuristics used by the server to match the examples are most effective when two or more facts are included in the query [2], due to limited information at the time we were unable to confirm that this is how developers would use the tool; this paper demonstrates that our assumption of large queries was valid.

4. Quantitative findings

By analyzing all of the saved interactions between the client and the Strathcona server, we were able to gain insight into how the developer used Strathcona during their query session. We analyzed three main types of data recorded by the Strathcona server.

Context queries. Context queries documents were sent from the client to the server whenever a developer selected some fragment of code and queried Strathcona. These documents contain a list of all of the structural facts comprising the query. These facts identify statically derivable method calls, field references, inheritance relationships, and type usages within the block of code the developer has selected.

Returned examples. Strathcona answers each context query with a set of structurally-relevant examples. Each example includes the same structural facts present in the context query so the Strathcona client can build a rationale explaining why the example is relevant for the developer's query and to build the UML representation of the example.

Source requests. If the developer deems an example interesting, they can request its source code; this document simply provides an identifier for the example the developer wishes to see the source code for.

The server records a timestamp for each of these documents, as well as a unique identifier for each host making the query. Unfortunately, we cannot tell from the server logs if the developer found an example useful or not; the only indicators we can use are whether the developer asked for the source code for an example. In this case, we infer that the developer felt the example could be relevant given its UML representation and assume the example was helpful in some way. We facted sessions as *successful* if they ended with a developer making a query and looking at the source code of at least one example (in contrast to ending with a query itself).

4.1. Session overview

Over the thirty-five month period of our Strathcona logs, 239 search sessions were initiated by 94 software developers (from at least 5 countries) encompassing 783 queries.[1] Figure 1 provides an overview of the number of context query and source requests made in each session. Developers averaged 2.4 context queries per session, although the median was only 1. They also requested 4.3 source examples on average, with a median of 2. 49% of sessions involved more than one context query, while 54% of sessions involved multiple source code requests. Figure 2 provides a breakdown of the 3652 query facts provided by developers.

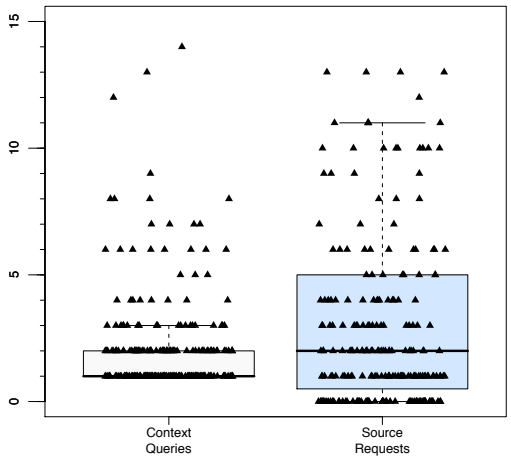
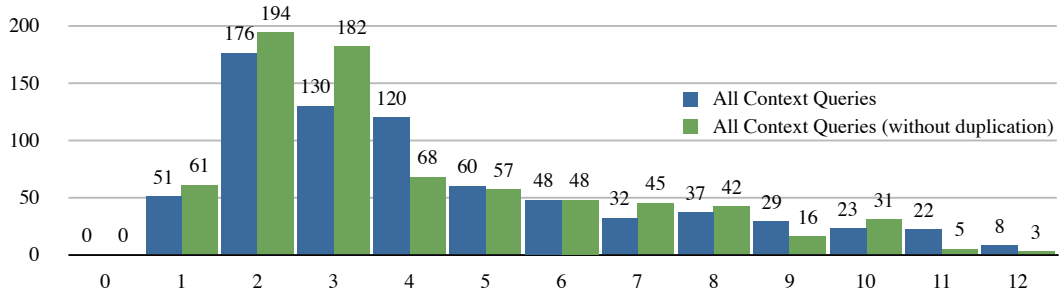
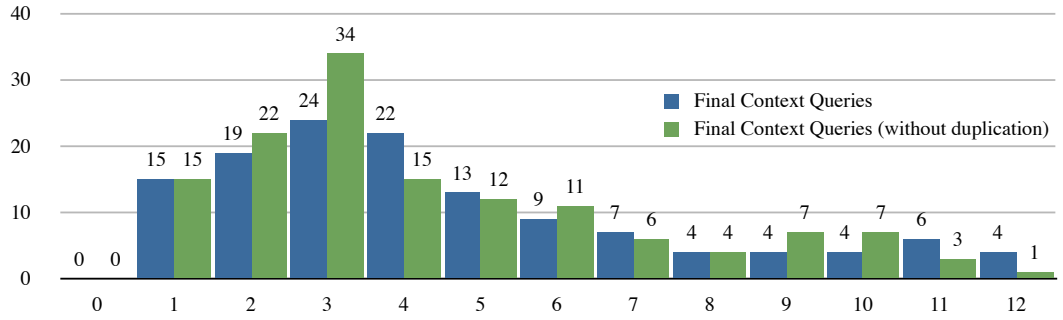


Figure 1. Number of context queries and source requests per Strathcona session.

[1] Before analyzing any of the data, we removed all of the sessions associated with our own usage of the Strathcona tool.



(a) Number of facts provided considering all Strathcona queries.



(b) Number of facts provided considering only the final query in a successful Strathcona session.

Figure 3. Each graph depicts number of structural facts included in a query (x axis) by the frequency queries of each size occurred (y axis). The top set of graphs consider all queries while the bottom pair consider only the final query of each session.

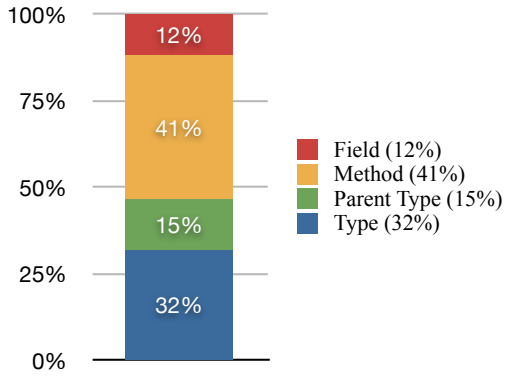


Figure 2. Proportion of query facts.

4.2. Queried facts

For this analysis, we combined all of the different kinds of search facts and treated them equivalently. Figure 3 provides a graphical representation of the queries; the x-axis represents the number of search facts while the y-axis represents the number of queries for each quantity of facts.

Strathcona considers a query fact representing a reference to `Status.OK` as two facts, the reference to the `Status.OK` field and a use of the `Status` class. To account for this, we provide both the total count of the facts as Strathcona interprets them as well as a version that does not duplicate any counts; we include both as different search approaches can chose to use one representation or the other. Figure 3(a) shows the number of facts for all of the context queries while Figure 3(b) shows the number of facts for only the final query of successful sessions.

In Figure 3(a) we can see that that while the median number of facts was two, developers provided three or more structural facts for 67% of their queries. For their final query (Figure 3(b)) the median number of facts has increased to three, with developers providing three or more structural facts for 74% of their queries. This clearly demonstrates that developers using Strathcona are formulating queries with multiple search facts and that they are adding facts to these queries as they progress through their search session.

While examining several sessions qualitatively, we found that while iterating on their query sessions developers were adding new facts to subsequent queries based on information present in example source code they viewed that was returned during prior queries.

5. Discussion

The internal validity of this study is hampered by the fact that Strathcona makes including additional search facts in a query trivial. While this is true, the interesting finding in

this paper is that the developers knew the facts to include in the first place. The external validity of our findings is limited from our lack of knowledge about the 94 developers who used Strathcona, if they were actually successful in finding the information they were looking for, and the obvious limitation of only having 239 search sessions to draw data from.

A key assumption of the Strathcona system was that developers would search for source code examples using multiple search facts; the more facts included in a query, the more effective Strathcona's heuristics tended to be [2]. This paper demonstrates that the assumption upon which Strathcona was created was valid and suggests that evaluations comparing Strathcona to other example recommendation systems should conduct their comparisons using several search facts in order to achieve a fair comparison of relative effectiveness.

Our analysis of the Strathcona usage data have given us several insights into how developers search for source code examples; these observations should be considered by researchers and practitioners creating source code search tools and services.

Developers search with multiple facts. Developers are able to elucidate multiple search facts when searching for context-relevant source code examples. This suggests that code search approaches should support and encourage searching using multiple terms; this can both help the developer to fully express their current knowledge and to constrain the result space to identify the most relevant examples possible.

Query sessions are iterative. Developers modify their queries over the course of a query session to specialize them as they identify new facts they deem relevant to their investigation; search tools should encourage iterative query refinement by including facilities that encourage developers to modify their queries and view their results in a lightweight manner. Tool designs that minimize the effort required to reformulate and specialize queries and reduce the effort required for the developer to glean useful facts from returned examples can help support iterative investigation.

Queries are composed of heterogeneous facts. While method calls were the most common kind of query facts, other types of facts were often included in queries. Facts relating to specific types made up 47% of queries (15% of these type facts related to parent classes and interfaces). Code search systems should enable developers to supply any kind of fact they are able to discern rather than forcing developers to only supply a single constrained kind of fact.

6. Conclusion

By analyzing 35 months worth queries sent to the Strathcona example recommendation system, we have found that developers predominantly queried Strathcona for source code examples using three or more search facts. We also found that as developers iterate on their searches, they tend to constrain their queries by adding more facts, as opposed to widening them by removing facts. These findings demonstrate that developers query for source code examples using multiple search facts in practice; this suggests that example recommendation tools should both allow developers to include multiple facts and make it easy for them to do so, enabling them to fully express the knowledge they have about the examples they are looking for.

Acknowledgements

I would like to thank David Notkin and Rylan Cottrell for their insight and assistance with this paper as well as reviewers for their comments and suggestions. This work has been funded in part through a NSERC Postdoctoral Fellowship.

References

[1] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 117–125, 2005.

[2] R. Holmes, R. J. Walker, and G. C. Murphy. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering*, 32(12):952–970, 2006.

[3] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the api jungle. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, pages 48–61, 2005.

[4] A. Michail. Data mining library reuse patterns using generalized association rules. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 167–176, 2000.

[5] S. Thummalapenta and T. Xie. PARSEWeb: A programmer assistant for reusing open source code on the web. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 204–213, 2007.

[6] T. Xie and J. Pei. MAPO: Mining API usages from open source repositories. In *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, pages 54–57, 2006.

[7] Y. Ye, G. Fischer, and B. Reeves. Integrating active information delivery and reuse repository systems. *SIGSOFT Software Engineering Notes*, 25(6):60–68, 2000.