Scaling up Evaluation of Code Search Tools through Developer Usage Metrics

Kostadin Damevski
Virginia State University
Petersburg, VA 23806
U.S.A.
kdamevski@vsu.edu

David C. Shepherd
ABB, Inc.
Raleigh, NC 27606
U.S.A.
david.shepherd@us.abb.com

Lori Pollock
University of Delaware
Newark, DE 19350
U.S.A.
pollock@udel.edu

***Abstract*—Code search is a fundamental part of program understanding and software maintenance and thus researchers have developed many techniques to improve its performance, such as corpora preprocessing and query reformulation. Unfortunately, to date, evaluations of code search techniques have largely been in lab settings, while scaling and transitioning to effective practical use demands more empirical feedback from the field. This paper addresses that need by studying metrics based on automatically-gathered anonymous field data from code searches to infer user satisfaction. We describe techniques for addressing important concerns, such as how privacy is retained and how the overhead on the interactive system is minimized. We perform controlled user and field studies which identify metrics that correlate with user satisfaction, enabling the future evaluation of search tools through anonymous usage data. In comparing our metrics to similar metrics used in Internet search we observe differences in the relationship of some of the metrics to user satisfaction. As we further explore the data, we also present a predictive multi-metric model that achieves accuracy of over 70% in determining query satisfaction.**

***Keywords*—code search, feature location, evaluation metrics, field studies**

I. Introduction

Developers performing software maintenance have to cope with increasingly complex systems and rapidly evolving code bases. Feature (or concern) location is a common activity at the outset of a software maintenance task, where a developer searches for an initial point in the code relevant to their maintenance task to begin their understanding of what needs to be changed. Developer studies have shown that code search consumes a substantial amount of time [1], while having an impact on the quality of the subsequent code changes [2]. To help developers, researchers have proposed and created a number of code search tools, based on a wide variety of different retrieval and corpus preprocessing algorithms. These code search tools commonly retrieve a ranked list of program elements to a user-supplied query [3].

The variety of proposed code search tools has prompted research in evaluation approaches that would enable understanding the effectiveness of the algorithms and guide future scientific advancement. Since controlled studies of developers in the field are expensive and difficult to perform at large scale, currently, most code search evaluation is performed using *gold sets* extracted from development histories of open source projects, consisting of pairs of bug reports and associated changed program elements [4]. The text or title of the bug report is used as a query in the code search tool, and metrics such as precision, recall, and accuracy are computed by comparing the result set retrieved by the code search tool to the program elements extracted from the change history.

While *gold set* evaluation is an invaluable prototyping tool in situ, field evaluations are necessary to ensure results from controlled studies are valuable to practitioners. Field data provides a broader data set reflecting the diversity of projects and purposes to which developers in the field may apply such tools. This type of evaluation extends the *gold set* both qualitatively, as developers may use a code search tool in unintended ways, and quantitatively, as the number of queries in a *gold set* is usually limited. It also helps to confirm or refute assumptions made in controlled experiments.

Unfortunately, evaluating a code search tool with field data presents its own set of challenges. Maintaining privacy and anonymity of actual field data is a requirement, which prohibits the recording and collection of source code or query strings. Maintaining good interactive response times during usage data collection is also a concern.

Our goal in this work is to address these concerns by enabling the evaluation of code search tools with empirical data collected from the field as developers perform their regular daily tasks, with the goal of scaling and transitioning these tools to increased developer use. Inspired by information retrieval evaluations [5], our approach is to infer user satisfaction with a search tool's retrieved results for a query by analyzing users' anonymized action streams. For instance, a user who executes a search and immediately opens a result is probably satisfied. A user who executes a search, browses several results without opening one, and then executes a modified search is likely not satisfied [5].

Specifically, this paper investigates a set of possible user satisfaction metrics that can be inferred from an anonymized activity stream, automatically collected by a deployed code search tool. The main contributions of this paper are:

- a proposed set of metrics and their implementation within the Sando code search tool [6] for inferring user satisfaction of code search results in the field,
- a controlled experiment to investigate which of these metrics correspond to user satisfaction,

- a field study on a larger data set, consisting of over 8000 user queries collected from Sando users in the field,
- the identification of a metric that correlates with user satisfaction in both user studies, and
- the construction of a predictive model, based on several metrics, with an accuracy of over 70% in predicting user satisfaction.

II. Related Work

This work is inspired by efforts and experiences in evaluating both Internet search and code search techniques.

A. Internet Search

Code search often uses similar principles as Internet search, by considering methods and classes to be documents that are indexed and searched. Our work was motivated in part by the prior work of Fox et al. [5], who performed a study to validate a set of browser evaluation metrics using explicit feedback collected from users of a modified Internet browser. We adapt and validate some of the metrics proposed by Fox et al. to the code search domain, specifically to code search tools that are implemented as extensions of an IDE. While the study by Fox et al. is larger in terms of number of participants and queries, we further validate our metrics on a realistic data set collected from code search tool users in the field.

Many others have focused on developing metrics to dynamically predict the effectiveness of a query (see [7] for a survey), to improve the retrieved result set. That work focused on recommending query reformulations, while we focus on post-hoc evaluation of code search tools, and develop satisfaction metrics calculated based on user interaction with a result set.

B. Code Search

A recent comprehensive survey of feature location techniques by Dit et al. [3] supports the motivation for this work, concluding that a major impediment to progress in feature location research is the difficulty in comparing approaches.

To address this, researchers have created Tracelab, a framework for rapidly prototyping and evaluating code search and search-related pre-processing analyses in a controlled experiment setting [4]. Code search algorithms can be rapidly constructed by composing a set of TraceLab components. Once such a TraceLab experiment is created, evaluation is conducted within the environment, using a set of provided gold sets. Our work is complementary to TraceLab by enabling evaluation in the field.

Others have proposed pre-retrieval evaluation metrics, commonly used to dynamically improve the results of a code search tool, for instance, by reformulating or expanding the user query [7]. Such metrics have previously been evaluated in the context of code search tools [8] and used for query expansion [9]. In this work, we focus on post-retrieval metrics because of their capacity to more accurately reflect the quality of a code search tool [7]. Although many of the metrics suggested for prediction may be of use for post-hoc evaluation as well, they are generally considered to be less accurate than the metrics that involve user interaction with retrieved results, such as the ones proposed in this paper. For instance, a high tf*idf score is less indicative of user satisfaction than the user actually looking at the result [7].

The only work we are aware of for evaluating code search techniques in the field with the goal of scalability is pairwise interleaving comparison of code search techniques as proposed in our prior work [10]. Code search techniques are compared by interleaving the result sets generated by both techniques unbeknownst to the user, with preferences determined by user clicks over time. While this approach was effective in improving the Sando code search tool, it is difficult to scale up, has considerable implementation costs, and can present annoyances to users by polluting the quality of the retrieved results to hide the presence of multiple techniques from the users. The work in this paper was motivated by these problems.

III. Metrics

Towards the goal of using user satisfaction feedback to evaluate code search tools in the field, we investigate several inexpensive metrics for inferring user satisfaction during daily use. We focus on developer usage metrics, i.e., metrics that are computed after the code search tool retrieves the results (which we refer to as the result set) to the user query and during the user's interaction with a result set. While such metrics have been used to improve Internet search for a decade [5], they have never been examined in the context of code search where the corpus (i.e. code) and results (i.e. program elements) may produce different user behavior patterns from Internet search.

The proposed usage metrics are available to any code search tool that has an active user base and can be gathered without any disturbance and with complete anonymity to its users. Once calculated, the metrics can be used to determine the effectiveness of the code search tool, (1) over all of the gathered field usage conditions, (2) for selected users or project types, or (3) for selected usage scenarios (e.g., to investigate specific query types that a code search tool may be poor at handling).

We list and describe the proposed set of code search evaluation usage metrics as follows:

- **Clicked Result Set.** Once a code search tool retrieves a result set to a given query, the user examines the results and subsequently either clicks or does not click on a result in the set. This metric is also known as clickthrough in Internet search. In IDE-based code search tools, a click usually corresponds to opening the retrieved result in the IDE editor. The binary metric Clicked Result Set indicates whether at least one click was made on any result in a given result set. The intuition is that clicked result sets are ones that may have satisfied the user's information need, while, conversely, unclicked result sets have failed to provide any results that the user wants to explore further.
- **Time on Result Set.** The time spent examining results can be an indicator of user satisfaction with that result set. However, it is unclear whether high or low values of this metric are best. For instance, result sets where the user quickly moves on to issue another query

may be dissatisfactory, while rapidly finding a desired result may be satisfactory. Both of these scenarios have similar values for this metric. Similar arguments can be made for high values of TIME ON RESULT SET; it is difficult to tell which is satisfactory and which is not. In the Internet search domain, high values of this metric are considered to be satisfactory [5], but this may not hold for code search.

- **Number of Clicks.** When the user clicks on a result set, they may click on any number of results in the result set, and even again on the same result after examining other results. For instance, they may click on one result in the retrieved set, investigate it as a starting point, and then come back and click on another result in the set for further exploration. As in the previous metric, more than one click on a given result set could indicate either a lesser or higher quality result set for the code search tool. For instance, more clicks on a given result set could indicate that the user wastes time in examining results in the retrieved result set that end up being fruitless, but in contrast, it could also be that the user is satisfying his or her information need with this process. For Internet search, it is expected that users are most satisfied when they quickly find a good starting point as one of the results in the result set, in which case, making only one click on the result set implies satisfaction.
- **Number of Short Clicks.** A short click is defined to occur when after clicking on a result, the user rapidly, within a short time span (usually less than 10-20 seconds), clicks on another result or issues another query to the code search tool. Short clicks, as well as their counterpart, long clicks, have been known to be used by Google to evaluate their Internet search engine [11]. We hypothesize that a short click in a code search tool, where a user spends only a small amount of time with a result, corresponds to dissatisfaction with that result. A number of such short clicks, in turn, reveals dissatisfaction with the entire result set.
- **Number of Long Clicks.** Long clicks, where a user spends a substantial amount of time (considered to be more than 30 seconds in Internet search) with a specific result before another interaction event can be viewed as representing a situation similar to when users keep relevant program elements in open windows in the IDE [1]. That is, a result set where the user spends a considerable amount of time, with at least one long click, indicates that the user believes that some of the results in the result set are relevant, implying positive user satisfaction with the result set. Similar to time spent on result set, this metric may have noise from the user closing the editor window to perform some other work and returning later to the code search tool, not necessarily indicating a long click.
- **Rank of Highest Clicked Result.** Eye-tracking studies of user interaction with search results in general have shown that most users examine result sets in a top-down fashion, where the highest ranked result (i.e.,the lowest rank value in a ranked list 1, 2, 3,...) receives attention first, followed by the second ranked, etc. [12]. If no promising results are observed, the user will give up on the entire result set after examining some number of the results. Consequently, we hypothesize that clicking on a higher ranked result (i.e., with a lower rank value) equates to relatively higher user satisfaction with the specific result set.

IV. EXPERIMENTAL DESIGN

The goal of this work is to validate whether this set of developer usage metrics can be used to evaluate the effectiveness of code search tools in the field, which would complement current *gold set* evaluation approaches. Specifically, we aim to answer the following research questions:

RQ1: What characteristics of the usage metrics correspond to developer satisfaction with a retrieved result set?

RQ2: Which developer usage metrics are most effective for evaluating code search?

RQ3: Can a statistical model based on the developer usage metrics be effective at code search tool evaluation?

To answer **RQ1**, we conducted an exploratory study where the participants provided explicit feedback on a Likert scale about their satisfaction with each result set retrieved by a code search tool. We compared participant satisfaction with the proposed developer usage metrics to determine how the two quantities relate (e.g. do high values or low values of a metric associate with developer satisfaction) as well as to choose good parameters for the long vs. short click threshold, necessary for the NUMBER OF SHORT CLICKS and NUMBER OF LONG CLICKS metrics.

To answer **RQ2** and **RQ3**, we used log data gathered from field usage of the Sando code search tool, consisting of large numbers of queries and users. Specifically, we leveraged queries that were part of a reformulation process, where the user modified an initial query by adding or removing terms, as indicators of high or low satisfaction. For **RQ2** we used these queries to determine whether specific developer usage metrics had the ability to differentiate satisfactory from unsatisfactory result sets (using hypothesis testing), and whether the metrics represented mutually independent information (using principal component analysis). For **RQ3**, we built a decision tree using the reformulated queries, which we cross-validated on the same dataset and tested on the explicit feedback dataset.

V. METRICS EXPLORATORY STUDY

We first performed a small-scale controlled exploratory study where we obtained explicit feedback from 17 users on their satisfaction rating of code search results. The explicit user ratings were used to learn how best to use the developer usage metrics, by relating the trends in the values of developer usage metrics with the explicit satisfaction feedback. We focused on which trends in values for time on result set and number of clicks on the result set relate to higher user satisfaction ratings. We also examined the relations between user satisfaction

ratings and the length of times between a click on result set and the next user interaction. This information provides insight into how to characterize a click as a short or long click.

A. Participants

To collect explicit user satisfaction feedback, we recruited 17 volunteers who were agreeable to complete a quick survey after each query result set was shown while they performed a predefined set of feature location tasks. The participants of this study were 17 computer science graduate students at Virginia State University and North Carolina State University enrolled in graduate software engineering courses. All participants have some programming experience of varying amounts.

B. Collecting Explicit User Feedback

We created a modified version of the *Sando* code search tool [6] such that it uses popup windows to ask the study participants to rate their satisfaction with the entire result set on a Likert scale, and with individual clicked results, as depicted in Figure 1.

Sando is an open-source code search tool[1] based on Information Retrieval (IR) search technology. It is built on top of the well-know Lucene [13] search library, which uses the vector space IR model with $tf * idf$ ranking. Sando includes several code specific techniques, including heuristics for weighting different parts of code, source code specific dictonaries for query recommendation, and supports several popular programming languages (e.g. C#, C, C++, and Java).

The survey for satisfaction with the entire result set is expressed in terms of how many results in the retrieved result set that the user believes are relevant to their most recent query. That is, the users could indicate that "None", "Few", "Some" or "Most" results were relevant. These were clarified by giving a quantitative match to each of the adjectives (e.g. "Few" corresponded to less than half of the presented results being relevant). We also offered the choice of *No opinion on the result set* on the popup to allow participants to opt out.

To survey users for satisfaction with an individual clicked result, the users were presented with a binary choice on the usefulness of a clicked result. In *Sando*, a click on a result opens the result in an editor window in the Visual Studio IDE. The feedback popup (bottom part of Figure 1) appeared after users had the time to evaluate the clicked result in the editor; it did not appear on every click, only periodically, to avoid annoying the participants. We used the feedback on individual clicked results to explore the amount of time for determining a long or short click.

C. Participant Tasks

To minimize effects of subjectivity, each participant performed the same 3 distinct feature location tasks extracted from real bug reports and feature requests in the *Family.Show* application [14], which is an open source genealogy application implemented in C#. We chose *Family.Show* as the subject project because it has retained a complete issue tracking history, which provides a number of issues representing realistic bugs and feature requests.

[1] http://sando.codeplex.com

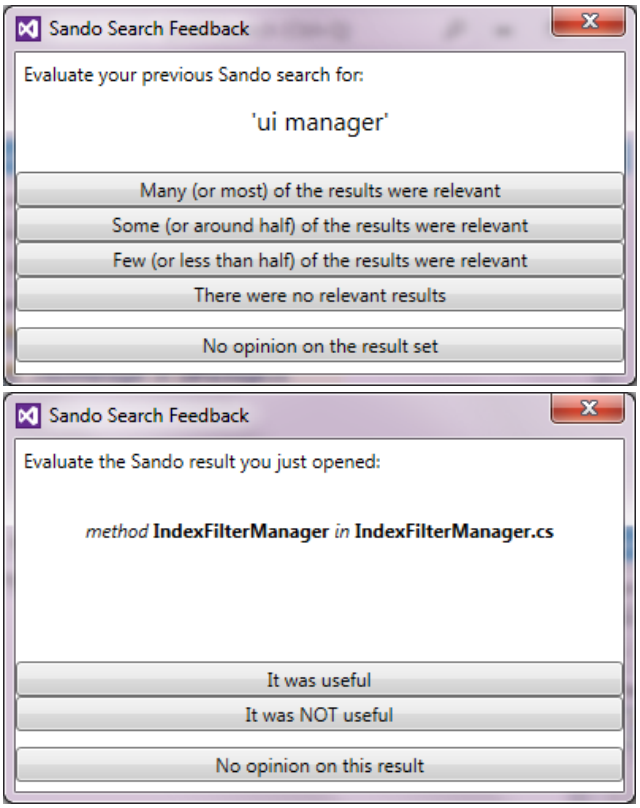

Fig. 1. Popups to gather participant feedback for entire result set (top), and a clicked result (bottom).

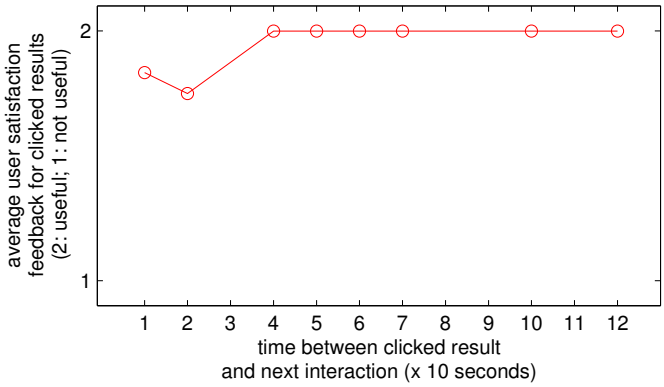

Fig. 2. Exploratory Study: Plot of explicit satisfaction feedback for individual results. Each point reflects an average of at least 3 clicked result satisfaction ratings. Average satisfaction differs most between the range below 20 seconds and the range above 40 seconds.

The issues used by the participants for feature location were chosen by reviewing the list of issues, starting with the most recent, and stopping when we identified three that met the following criteria: (1) the issue was self-contained and written in a way that a developer with limited knowledge of *Family.Show* could understand (e.g., no references to uncommon family tree data formats); (2) the issue was filed by a user, indicating that it was realistic. The issues we chose were numbers 949, 429, and 455, and are available online[2].

[2] http://familyshow.codeplex.com/workitem/list/basic

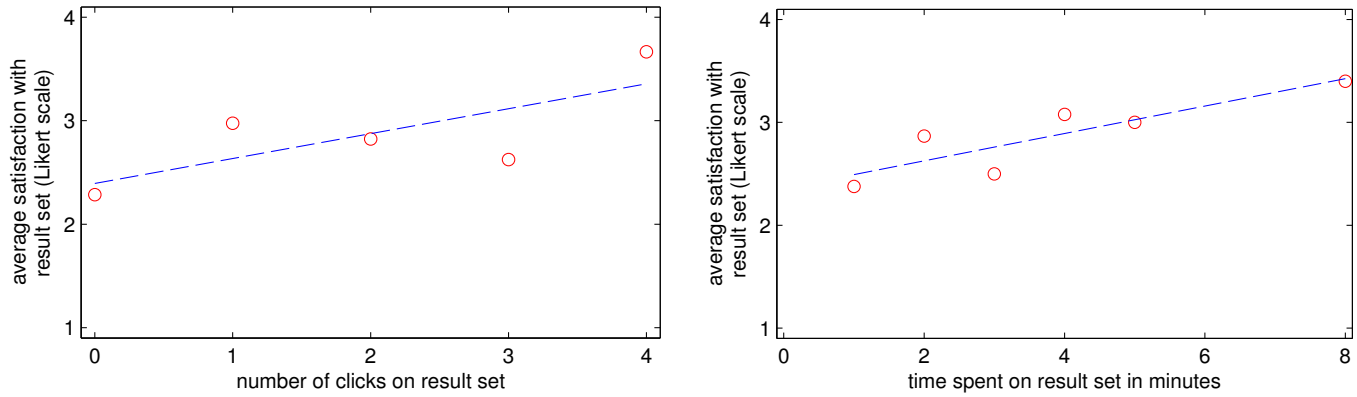

Fig. 3. Exploratory Study: A plot of developer usage metrics, NUMBER OF CLICKS (left) and TIME ON RESULT SET (right), to user satisfaction represented by a Likert scale of 1-4. Each plot includes data points representing average satisfaction for a specific metric value with at least 5 data points and a regression line.

D. Procedure

Before being asked to perform the tasks, the participants were shown a demo of *Sando*. They also received a demonstration of the capabilities of the subject application, *Family.Show*. In addition, an example of a feature location task, which used a separate bug report, was performed by the study moderator to further demonstrate what was expected from the participants.

The participants were given the modified version of the *Sando* code search tool, which displayed the popup window asking user satisfaction feedback after each search. Given the summary of a bug report or feature request, consisting of a few declarative sentences, the participants located an initial starting point in the code for the task using the *Sando* code search tool and the Visual Studio IDE. The participants were asked to stop once they were satisfied with what they had found, or after a 15 minute per-task time span had passed.

The popup was displayed either after a timespan of 5 minutes had expired since the query was issued or when a participant began to reformulate the query. Logs collecting information about the participants' interaction with *Sando* were also continuously collected during the tasks. These logs contained enough information to compute, for each query, the set of effectiveness metrics of our study.

E. Study Results

To choose a threshold for a click to be considered a long or short click, we identified individual results that were clicked by at least 3 users, averaged their satisfaction ratings, and then plotted the averages against the length of time between the clicked result and the next user interaction. Figure 2 presents the data plot showing the range of averages between 1: 'NOT useful' and 2:'useful'.

While overall satisfaction feedback tended to be mostly positive (2 or close to 2) across all responses, the averages are much less in the 0-20 second range than above the 40 second mark, when the averages become completely positive (equal to 2). Based on this observation, we set the thresholds as 0-20 seconds for a short click and 40 seconds and above for a long click. Since there were no data points in the in-between range of 20-40 seconds, we consider this time interval as one where it is likely too ambiguous to differentiate between short and long clicks. In practice, while having this in-between interval will likely reduce the number of data points that we can characterize as short or long clicks, it should raise the fidelity of the long/short click characterization. This is important as long clicks have a strongly positive while short clicks a strongly negative connotation with respect to user satisfaction.

To characterize which trends in values for TIME ON RESULT SET and NUMBER OF CLICKS on the result set relate to higher user satisfaction ratings, Figure 3. Using the slope of the linear regression line, we determine that high values indicate higher user satisfaction ratings for both TIME ON RESULT SET and NUMBER OF CLICKS metrics.

We summarize the results of our observations, for all of the proposed developer usage metrics, in Table I. This table, including the threshold values for short and long clicks, serves as an answer to **RQ1**.

TABLE I. METRICS PROPOSED TO INFER SATISFACTION.

Metric	*Indicator of High Satisfaction*
CLICKED RESULT SET	Clicked
TIME ON RESULT SET	High value
NUMBER OF CLICKS ON RESULT SET	High value
NUMBER OF SHORT CLICKS	Low value
NUMBER OF LONG CLICKS	High value
RANK OF HIGHEST CLICKED RESULT	Low value

F. Threats to Validity

Since the participants needed to be recruited and agreeable to providing user feedback after each query, the study is small, consisting of the result set satisfaction feedback of 17 participants on three time-limited tasks. The threat presented by the small study size is mitigated by the type of research questions that we aim to answer using the collected data, which are appropriate for this limited data set.

Another threat is the range of participant experiences with development in general and code search in particular, influencing different types of search strategies that could have been employed during the study. Yet another threat comes from the fact that while the chosen tasks were unrelated, exploring separate parts of the code base, there could have been some

participant learning between them which could have biased the outcomes. Both threats are mitigated by two factors: 1) the overall positive nature of the participants' satisfaction scores is indicative of a high rate of search session success both over time and across participants; and 2) the proposed metrics are reflective of search satisfaction with each retrieved result set in isolation, reducing the effect of search strategies that span more than one query.

VI. Usage Metrics and Search Satisfaction in the Field

To overcome the limitations of the small-scale study and strengthen the investigation into the potential use of developer usage metrics for evaluating code search tools, including addressing **RQ2** and **RQ3**, we conducted a field study of the developer usage metrics on a larger scale through anonymous participation with indirect user satisfaction information collected.

We collected data for the metrics computations from unknown code search tool users who opt-in during tool installation to anonymized data collection (with no user survey interruption). The collected data reflects developer behavior when using the tool during their normal daily tasks, recorded as a stream of anonymously collected events for computing usage metrics reflecting limited aspects of the query and corpus.

Specifically, we exploited the extensibility and existing global user community of the Sando code search tool [6] to create a prototype of our strategy for automatically-collected, anonymous field usage data. Sando is currently distributed mainly via its Visual Studio Gallery site[3], where it has gathered several thousand downloads since its initial release.

A. Usage Data Collection Methodology

We modified the Sando search tool to generate a log of timestamped events corresponding to user activity (e.g., when a user submits a query or clicks on a result in a retrieved result set). Each logged event is further attributed with properties specific to the event (e.g., the number of terms in the query, the number of retrieved results). Table II depicts the types of events we log and their associated properties, to gather usage data for computing the usage metrics.

The primary requirements in enabling a code search tool to collect usage data from those who opt-in include, (1) maintaining low performance overhead, and; (2) protecting user anonymity and privacy while maximizing the avenues of use for the collected data. This section describes how we address these challenges.

1) Low-Overhead Data Collection: To maintain good tool-user interactivity, we minimized the performance overhead in several ways. The event log is stored on the user's machine during their daily interaction with the Sando search tool and periodically transmitted to a central server. Sando uploads the logs to the server at opportune times, when fast user response time is not critical, but often enough to ensure their size does not reach proportions where their upload could observably slow down the system. We used the Amazon S3[4] cloud storage service as a means to collect these logs from all Sando users, due to its relatively low cost and easy setup.

TABLE II. Logged Usage Data Events in Sando.

Event	*Properties*
Solution opened	hash of solution name
Query recommendation used (if any)	small type of recommendation (pre or post search, spelling, corpus term), rank of recommendation
Submitted query	number of terms, each term's type (camelcased, plain, undercore, abbreviation), term-level similarity to previous query
Sando retrieved results	number of results, average ranking score, std. deviation of ranking scores
Single or double-click on result	rank of result, type of program element (e.g. class, method), ranking score

We use asynchronous (nonblocking) communication with a low-priority background thread to perform the I/O for the log messages. Many high-level languages now offer efficient event (publish subscribe) mechanisms that are appropriate for this purpose. Specifically, our implementation of usage data collection for Sando utilizes the capabilities of the Log4Net[5] tool to persist the logs. We also exploited the common capability to centralize the log message generation to one component in the code, which allows for consistency between log messages especially as the code base evolves and new log messages are added.

2) User Privacy and Anonymity: Privacy represents the need of users to keep certain information to themselves, which often corresponds to the content of the source code and the content of the queries in code search tools. These items can often reveal proprietary information that should be excluded from any usage data collection. A tool that exposes any private information would likely be banned from use during professional development at all companies. Sando's usage data collection does not include any private information. For instance, we do not record any file names or user query strings, even though this information would be helpful for furthering code search research.

Anonymity, on the other hand, represents the need of users not to be easily recognized from the data that is recorded by the code search tool. For this purpose, we typically mask information that may identify a specific user using a hash function that is difficult to reverse engineer. For instance, in Sando's usage data collection, we record the user's machine name to identify a specific user for analysis of his or her individual search patterns, but we obscure the name via a hash function.

The collected metrics in Table II are defined to gather properties without sacrificing the user privacy and anonymity.

B. Measuring User Satisfaction

Collecting usage data from the field of regular users of a tool cannot rely on direct user feedback through popup survey windows that can annoy users. Thus, to enable a large-scale study of the potential of the usage metrics for code search evaluation we use a feature of the submitted queries as an indication of user satisfaction in our study.

[3] http://visualstudiogallery.msdn.microsoft.com
[4] http://aws.amazon.com/s3/
[5] http://logging.apache.org/log4net

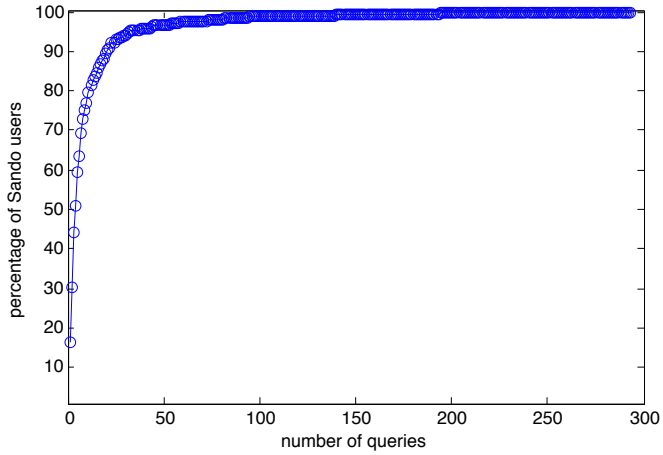

Fig. 4. Cumulative distribution function of the percentage of Sando users issuing a certain number of queries.

Most models of information-seeking with information retrieval tools developed via extensive observational studies describe search as an iterative process of successively refining an initial query until the user's information need has been met [15], [16]. In some instances, the user retains parts of the initial query and reformulates it by either removing terms (which broadens the results) or adding terms (which focuses the results). This reformulation occurs due to the inadequacy of a retrieved result set for the specific information need, which is indicative of user dissatisfaction with that result set. Thus, the queries that were subsequently reformulated can be used as an indicator of negative user satisfaction with the result set just before the reformulation (pre-reformulation).

In contrast, queries on which the reformulation process terminated (which we call post-reformulation queries) have likely satisfied the user's information need, which indicates satisfaction with the retrieved result set; otherwise, the reformulation process ought to have continued. While it is possible that the user decided to give up before finding a suitable result, we believe that this scenario occurs rarely. Since reformulation requires effort and planning from the user, he or she must believe the process will result in successful retreival, unlikely wasting effort reformulating if the likelihood of failure is high. Further evidence of post-reformulation queries indicating user satisfaction with the result set was observed in the relative lengths of the post-reformulation queries versus all queries in the *Sando* dataset. Other studies have shown that longer queries are more satisfactory [17]; our data set shows that post-reformulation queries consisted of 2.5 terms on average compared to 1.3 terms across the entire *Sando* dataset.

Thus, we use query reformulation categorization to indicate user satisfaction as: queries that were subsequently reformulated indicate result sets that were not satisfactory, while result sets of queries that resulted from the reformulation, on which the reformulation process terminated, are are considered satisfactory. To automatically identify reformulated queries in our field usage data, we computed the lexical similarity between two consecutive queries[6], to detect when a user has added or removed terms to the previously issued query.

Since the number of reformulated queries is small compared to the overall dataset (i.e. less than 1 in 10 queries in our data set were part of a reformulation strategy) it cannot be directly used to evaluate code search tools at the individual query level. On the other hand, the metrics proposed in this paper have the ability to be used at any level of evaluation granularity.

C. Study Results

In total, we collected usage data over 8164 queries by developers in the field during their use of the Sando code search tool. Data collection lasted over a period of 9 months and included 709 individual Sando users who used the tool on an average of 1.57 separate software projects[7].

To provide some insight into the user population, Figure 4 shows the percentage of the Sando user population that has issued a specific number of queries or less over the usage data collection period. The average number of queries per user was 9.63. While 20% of the users issued only only one query, super-users generated 100-250 queries using the tool. The reformulated query portion of this field data set consisted of 398 queries that were issued and subsequently reformulated and 274 queries that resulted from the reformulation; a total of 672 queries.

Table III shows average values for each metric when result sets were for queries pre-reformulation versus post-reformulation, including a relevant test statistic and the p-value indicating the statistical probability of each metric being independent of result set satisfaction. For instance, 30% of all post-reformulation queries' results had at least one click on the result set, while only 16% of queries with pre-reformulation queries had at least one click on the result set. The average time spent on result sets of post-reformulation queries is 1130 seconds (or over 18 minutes), while the average time on pre-reformulation queries was only 252 seconds (or just over 4 minutes). The average number of clicks on a result set over all queries for post-reformulation queries was 0.66, and 0.25 on pre-reformulation queries. The RANK OF HIGHEST CLICKED RESULT metric can be computed only on reformulation queries that also have at least one click (of which there are 147 total, 64 pre-reformulation and 83 post-reformulation), as a value for RANK OF HIGHEST CLICKED RESULT on unclicked queries does not exist.

To answer **RQ1** and **RQ2**, we evaluated whether the null hypothesis that each of the metrics does not significantly differentiate satisfaction, as expressed by whether a query was pre or post reformulation, using the chi-squared test of independence for the nominal metric CLICKED RESULT SET and a one-sided Mann-Whitney U Test on the remaining metrics. The one-sided test was used with the trend for each metric that was initially observed in the exploratory study (e.g. that high values of NUMBER OF CLICKS were better). Based on a p-value threshold of less than 0.05, there are several

[6]We use term-level Dice similarity between two consecutively submitted queries.

[7]This was identified by the hash of each user's machine name, machine domain, and project name.

TABLE III. RELATIONSHIP OF METRICS TO REFORMULATED VS. NON-REFORMULATED QUERIES. METRICS THAT EXHIBIT A STATISTICALLY SIGNIFICANT RELATIONSHIP (P ¡ 0.05) ARE STARRED.

Metric	*Pre-Reformulation Query (Not Satisfied)* N = 398	*Post-Reformulation Query (Satisfied)* N = 274	*Test Statistic*	*p-value*
CLICKED RESULT SET*	16%	30%	X^2 = 18.35	0.000
TIME ON RESULT SET*	252s	1130s	W = 61141.5	0.003
NUMBER OF CLICKS*	0.25	0.66	W = 62572	0.000
NUMBER OF SHORT CLICKS	0.08	0.14	W = 56495	0.971
NUMBER OF LONG CLICKS*	0.01	0.07	W = 57032.5	0.000
RANK OF HIGHEST CLICKED RESULT ▷ clicked queries	 N = 64 4.71	 N = 83 5.30	 W = 2709	 0.587

of the metrics that show statistically significant differences in satisfaction: CLICKED RESULT SET, TIME ON RESULT SET, NUMBER OF CLICKS, and NUMBER OF LONG CLICKS. The NUMBER OF SHORT CLICKS metric had very high p-values on the one-sided test indicating a strong relationship in the opposite side of the test, which is further suggestive of the strong positive influence of a click on the result set, regardless of the time the user lingers on the individual opened result. Thus, the larger-scale study confirmed the observations of trends for the usage metrics with regard to user satisfaction in the smaller study.

D. Metrics Redundancy

To determine whether each of the proposed code search evaluation metrics is unique or whether some of them collectively support a specific underlying dimension of behavior in the searchers, we applied Principal Component Analysis (PCA) to the Sando dataset. The results, consisting of 4 components as shown in Table IV, constitute 87% of the variance in the dataset. The bolded values represent values greater than 0.5, which we look to when interpreting the PCs. In the bottom rows of Table IV, we show the proportion of the variance explained by each PC as well as the cumulative variance for the PCs up to that point.

TABLE IV. RELATIONSHIP OF METRICS TO REFORMULATED VS. NON-REFORMULATED QUERIES.

Metric	PC_1	PC_2	PC_3	PC_4
CLICKED RESULT SET	**0.51**	0.24	0.06	0.04
TIME ON RESULT SET	0.15	0.26	**0.90**	0.30
NUMBER OF CLICKS	**0.56**	0.27	0.07	0.06
NUMBER OF SHORT CLICKS	**0.51**	0.33	0.14	0.36
NUMBER OF LONG CLICKS	0.32	0.03	0.24	**0.88**
RANK OF HIGHEST C.R.	0.21	**0.83**	0.33	0.02
Proportion	39%	17%	16%	15%
Cumulative	39%	56%	72%	87%

The results of PCA analysis support the conclusion that all of the metrics are fairly unique in representing a dimension of user behavior. The results indicate that CLICKED RESULT SET, NUMBER OF CLICKS, and NUMBER OF SHORT CLICKS are the most correlated; however, this correlation is not extremely strong, indicated by relative weakness in the representative factors of PC_1: 0.51, 0.56 and 0.51. The remaining metrics, i.e. RANK OF HIGHEST CLICKED RESULT, TIME ON RESULT SET, and NUMBER OF LONG CLICKS, each dominate one specific PC, which is indicative of their independence, while the corresponding PCs (2 through 4) contribute a similar amount to the variance in the dataset, ranging from 15% to 17%, indicative of a even distribution of the discriminating power of each of these metrics.

E. Predictive Model of User Satisfaction

RQ3 focuses on whether a statistical model based on developer usage metrics could be effective for code search tool evaluation. To build a predictive model of user satisfaction with code search, that may use combinations of the metrics, we require a training set of satisfactory/unsatisfactory queries. For this purpose, we turn again to the reformulation characterization of queries in the Sando dataset, which represent user satisfaction or dissatisfaction with a retrieved result.

Using this dataset, which consists of a total of 672 queries, we learned a decision tree using the J.48 algorithm, which minimizes the information loss at each level of the tree. A decision tree as a classification tool has the added benefit of its results being easily interpreted by a human.

The learned decision tree is shown in Figure 5. It has 70% prediction accuracy, which was computed using 10 fold cross-validation. The first level of the tree uses the TIME ON RESULT SET metric, classifying result sets with $<= 3$ seconds and those with > 66 seconds as satisfactory. The remaining result sets are classified by the second level of the decision tree, which utilizes the CLICKED RESULT SET metric, to be unsatisfactory if unclicked. The third level of the decision tree classifies the remaining result sets using the NUMBER OF CLICKS metric, where those that had less than or equal to 5 clicks are predominantly satisfactory. Also, the learned decision tree is better at predicting result set dissatisfaction than satisfaction, where it encounters a large proportion of false positives on the test set.

To further test its efficacy, we applied the decision tree on the data gathered in the exploratory study. As a baseline, the average Likert score for result sets in the exploratory dataset was 2.75 out of 4. For the result sets predicted to be satisfactory, the average Likert score was above the average at 3.07, while for those predicted to be unsatisfactory the Likert score was below average at 2.35. If we assume responses of the popup of "Many", "Some" and "Few" to be indicative of satisfaction, the prediction accuracy is 87%, while if we consider only "Many" and "Some" to be satisfactory, the accuracy drops to 74%. Since both of these accuracy levels are high enough to make this model usable for evaluating code search tools, we can answer **RQ3** affirmatively.

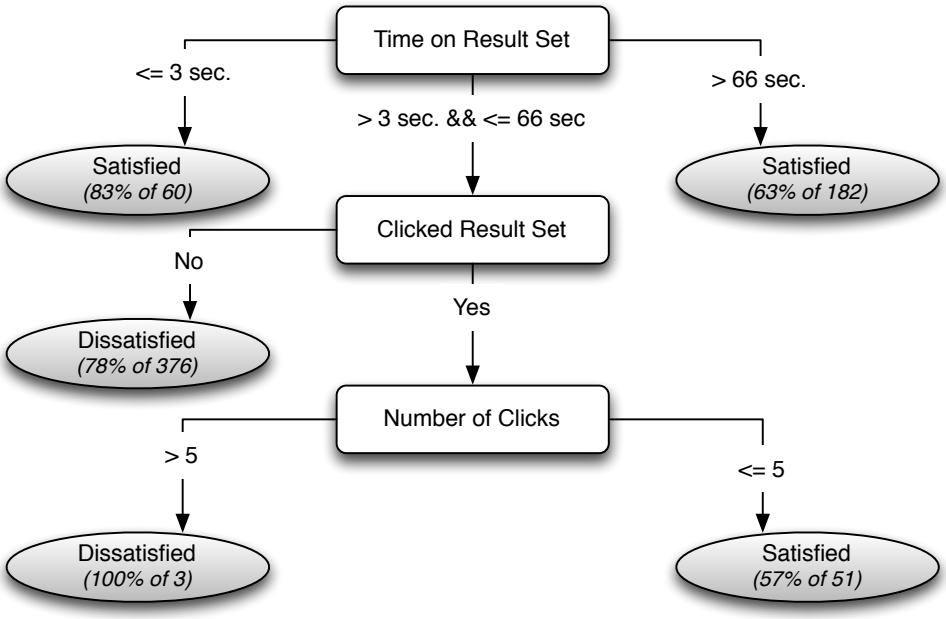

Fig. 5. Decision tree to predict result set satisfaction, learned from the Sando dataset.

F. Threats to Validity

The main threat to validity of this study comes from the source of the ground truth dataset, consisting of reformulated queries. It is possible that such queries do not represent the characteristics of most queries issued to *Sando*. This is mitigated by the fact that reformulation is a common strategy in search, corresponding to the accepted "berry picking" model of human behavior in this task [18]. The reformulated dataset also represents many different developers using the *Sando* search tool in their daily work.

Another threat comes from the use of only one code search tool - *Sando*. This threat is mitigated by the fact that *Sando* has capabilities representative of most other available code search tools, such as being based on the popular vector space model, using word stemming to match words with the same root, and supporting query recommendation based on artifacts in the code base.

VII. Discussion

In this section, we discuss the implications of our findings in two respects: as a means of performing code search evaluation and as they relate to existing knowledge in Internet search evaluation.

Several of the metrics that we propose had some correlation to developer satisfaction with a result set. The decision tree extracted from our dataset also indicates that several metrics are useful in discerning satisfactory from unsatisfactory result sets. Two of the metrics Clicked Result Set and Time on Result Set were strongest, both in their individual association to satisfaction and in their role in the decision tree. Out of the metrics we examined, these two metrics are highly likely to produce relevant evaluation of code search tools in the field. The prominence of these two metrics also indicates that most of the time, developers are discerning searchers that fairly quickly judge a result set as satisfactory or not and do not waste clicks or time on unpromising results.

The metrics Number of Clicks and Number of Long Clicks also appeared to be valuable to code search evaluation by classifying a few corner cases of developer behavior. Number of Clicks is useful for the few cases where too many clicks yielded dissatisfaction with a result set. Number of Long Clicks provided a more discerning way of counting individual clicks, as when a user spends more than 40 seconds, with an open result we can be very confident of his or her satisfaction. This metric, however, was relatively sparsely available in the dataset, which hindered its larger role in the decision tree. The remaining metrics, Number of Short Clicks and Rank of Highest Clicked Element, were not good predictors of satisfaction in our study. While we expected a larger number of Number of Short Clicks to relate to dissatisfaction with a result sets, we observed that most of the time a developer click was a good indicator of satisfaction, regardless of the short period of time. A similar observation can be made for Rank of Highest Clicked Element; the rank just did not matter enough to users.

It is intuitive that the relevant values of the developer usage metrics we describe in this paper might differ between Internet search and code search, because of the different time requirement and cognitive load of evaluating a retrieved result (i.e. an Internet page vs. source code) are likely very different. Some of the metrics were indeed differently characterized in code search than Internet search evaluation. For instance, low values of Number of Clicks are considered best in Internet search, while in code search finding several relevant program elements in our study was generally beneficial up to a point. In Internet search, high values of Time on Result Set are considered indicative of poor satisfaction, while in our study of code search, higher values were always beneficial. This can be attributed to developers being happy with longer

result examination time as a broader program comprehension strategy.

The learned decision tree also shows that fast lookups are common in code search and developers are happy when quickly finding what they are looking for. This further confirms observations by many others of two types of queries in code search: those where developers lookup a known program element, which are best answered quickly and with few clicks, and those where developers are trying to find and comprehend a feature [10], [19], which are best answered via many relevant program elements.

VIII. Conclusions and Future Work

In this paper, we investigated whether automatically gathered anonymous usage data from code searches in the field can be used to compute post-retrieval metrics that can then be used to infer user satisfaction. If so, they could be used to evaluate a code search tool post hoc from the perspective of a working developer's use. Both privacy and user anonymity are preserved. We designed and conducted both a controlled study and a larger field study to analyze the potential correlation between individual post-retrieval metrics with user satisfaction of a code search tool under different usage.

Our results indicate that several metrics, namely clicking on a result set, time spent on a result set, number of clicks and number of long clicks are all correlated with statistical significance with user satisfaction. We also found, via principal component analysis, that each of these metrics is fairly independent of one another, measuring separate underlying aspects in our data set. A decision tree constructed from these metrics has an acceptable classification accuracy. Analysis of the strongest branches of this decision tree indicates that users are satisfied when they find the result quickly (within 3 seconds of the query) or when they spend a substantial amount of time interacting with the retrieved result set (over 66 seconds). Also, when the result set is neither of the above and also does not receive a click, then it is highly likely that the user is unsatisfied.

The data for these metrics can be collected through adding logging capabilities to the code search tool and the metrics can be computed while maintaining privacy and anonymity and without hurting interactive response times. These results provide a step toward evaluating code search tools based on field usage in a scalable way, which will help to transition them into practical use.

More work is needed to investigate how other types of metrics, derived from the corpus or query, could enhance the evaluation of code search tools in the field. We are also investigating other possible developer-centric metrics that can be computed while achieving the goals of metrics gathered in the field.

References

[1] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. on Soft. Eng.*, vol. 32, no. 12, pp. 971–987, 2006.

[2] M. Robillard, W. Coelho, and G. Murphy, "How effective developers investigate source code: an exploratory study," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 889–903, 2004.

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Soft. Maint. and Evolution: Research and Practice*, 2011.

[4] B. Dit, E. Moritz, and D. Poshyvanyk, "A tracelab-based solution for creating, conducting, and sharing feature location experiments," in *IEEE Int. Conf. on Program Comprehension*, 2011.

[5] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White, "Evaluating implicit measures to improve web search," *ACM Transactions on Information Systems*, vol. 23, no. 2, pp. 147–168, Apr. 2005.

[6] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: an extensible local code search framework," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE, 2012, pp. 15:1–15:2.

[7] D. Carmel and E. Yom-Tov, *Estimating the Query Difficulty for Information Retrieval*. Morgan and Claypool, 2010.

[8] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Automatic query performance assessment during the retrieval of software artifacts," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE. New York, NY, USA: ACM, 2012, pp. 90–99.

[9] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *International Conference on Software Engineering (ICSE)*, 2013.

[10] K. Damevski, D. Shepherd, and L. Pollock, "A case study of paired interleaving for evaluating code search techniques," in *Proceedings of the IEEE Conference on Software Maintenance and Reengineering - Working Conference on Reverse Engineering (CSMR-WCRE)*, 2014.

[11] S. Levy, *In the Plex: How Google Works, Thinks, and Shapes our Lives*. Simon and Schuster, 2011.

[12] L. Granka, T. Joachims, and G. Gay, "Eye-tracking analysis of user behavior in www search," in *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2004, pp. 478–479.

[13] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Greenwich, CT, USA: Manning Publications Co., 2010.

[14] Family.Show Geneology Application, "`http://familyshow.codeplex.com`."

[15] A. Sutcliffe and M. Ennis, "Towards a cognitive theory of information retrieval," *Interacting with computers*, vol. 10, no. 3, pp. 321–351, 1998.

[16] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[17] N. J. Belkin, D. Kelly, G. Kim, J.-Y. Kim, H.-J. Lee, G. Muresan, M.-C. Tang, X.-J. Yuan, and C. Cool, "Query length in interactive information retrieval," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, ser. SIGIR '03. New York, NY, USA: ACM, 2003, pp. 205–212. [Online]. Available: http://doi.acm.org/10.1145/860435.860474

[18] M. J. Bates, "The design of brosing and berrypicking techniques for the online search interface," *Online Information Review*, vol. 13, no. 5, pp. 407–424, 1989. [Online]. Available: http://ci.nii.ac.jp/naid/80004823012/en/

[19] S. K. Bajracharya and C. V. Lopes, "Analyzing and mining a code search engine usage log," *Empirical Softw. Engg.*, vol. 17, no. 4-5, pp. 424–466, Aug. 2012.