On Understanding How Developers Use the Spotter Search Tool

Juraj Kubelka*, Alexandre Bergel*, Andrei Chiş †, Tudor Gîrba†, Stefan Reichhart†, Romain Robbes*, and Aliaksei Syrel†

*PLEIAD Laboratory, Department of Computer Science (DCC), University of Chile, Santiago, Chile

Email: {jkubelka, abergel, rrobbes}@dcc.uchile.cl

†Software Composition Group, University of Bern, Switzerland

Email: andrei@iam.unibe.ch, tudor@tudorgirba.com, stefan.reichhart@gmail.com, aliaksei.syrel@students.unibe.ch

***Abstract*—Analyzing how software engineers use the Integrated Development Environment (IDE) is essential to better understanding how engineers carry out their daily tasks. SPOTTER is a code search engine for the Pharo programming language. Since its inception, SPOTTER has been rapidly and broadly adopted within the Pharo community. However, little is known about how practitioners employ SPOTTER to search and navigate within the Pharo code base.**

This paper evaluates how software engineers use SPOTTER in practice. To achieve this, we remotely gather user actions called events. These events are then visually rendered using an adequate navigation tool chain. Sequences of events are represented using a visual alphabet.

We found a number of usage patterns and identified underused SPOTTER features. Such findings are essential for improving SPOTTER.

I. INTRODUCTION

Integrated Development Environments (IDEs) are standard environments to develop software applications, *e.g.,* Eclipse, Xcode, PHARO IDE. IDEs come with a collection of features that facilitate change tasks. Understanding which features are used and how they are being used is an important research question; for example refactoring tools automate source code manipulation, but studies reveal that they are seldom used [1].

SPOTTER is a search tool for the PHARO IDE that supports flexible ways to search for information in a source code base. Nonetheless, creating an efficient tool requires knowing how users employ the tool and whether or not they use it at its fullest potential. For the purpose of this analysis, we form a visual alphabet and an interactive and tailored software visualization—as part of a tool called SPOTTER ANALYZER—that aim to answer the following research questions:

- ***RQ1.*** *Are there any missing* SPOTTER *features?*
- ***RQ2.*** *Are there any underused* SPOTTER *features?*

Outline. Section II introduces the SPOTTER search tool. Section III gives the terminology for the collected data and presents the SPOTTER ANALYZER tool. Section IV discusses the observed user behavior, and reveals missing and seldom used SPOTTER features. Section V explores related work. Section VI summarizes our work.

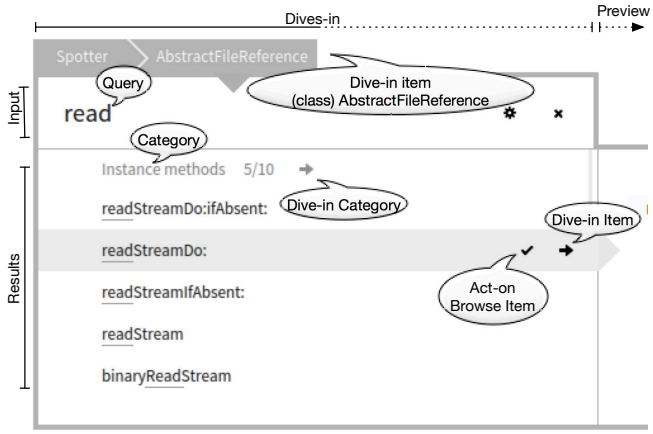

Fig. 1. SPOTTER UI is composed of an input field for queries, results, preview, and dives-in bar that indicates where a user searches for. The results are structured into categories. The user can dive-in a category or a selected item, or can act-on the selected item (by pressing the enter key) that usually opens the selected item in a specific tool.

II. SPOTTER

SPOTTER[1] is a moldable development search tool implemented in the PHARO IDE and introduced to the community in December 2014. SPOTTER searches through a wide range of software entities—*e.g.,* code, objects, documentation—and navigates through various dependencies, *e.g.,* method calls, class references. Figure 1 illustrates a search session.

Category. SPOTTER crawls over more than one hundred sources of information, *e.g.,* classes, methods, menu items, recently modified packages, recently written scripts, or help topics—such information sources are called *categories*. Users can customize what information to search for and how to present it in the SPOTTER view.

Category filter. When a user enters a query, SPOTTER searches through and displays multiple types of results. For example, if a user enters `read`, SPOTTER searches through instance variables, methods, classes, *etc.* The output can be filtered out by a category name that begins with the # character; we call it *category filter*. If a user wants to search only for instance

[1]http://scg.unibe.ch/research/moldablespotter

methods that contain the word `read`, the query has to be formulated as `read #i` or `read #instance`.

Dive-in item. After selecting an item, the user can *dive-in item* and search for information related to that item. Figure 1 outlines a scenario where a user selected the `AbstractFileReference` class, dived-in the class, and is searching for `read` inside the class—the figure shows instance methods belonging to the class that matches the query. A user can also look for possible information in the class such as variables, references, superclasses, subclasses, packages, etc.

Dive-in category. To avoid presenting a long list of candidate results, only the first five results for each category are displayed. Figure 1 displays five out of ten existing instance methods that contain the word `read`. A user can display all results using the *dive-in category* action attached to the "Instance methods" category.

Use case. Both dive-in item and dive-in category features make source code exploration and navigation more accurate. For example, if a user wants to read a file, first the user can search for a class that contains the word `file` and then dive-in a particular class and search for methods of that class that contain the word `read`. In addition, the user can dive-in an interesting method and observe information relevant to the method.

***Open questions with* SPOTTER.** While SPOTTER developers perceive the tool usage as convenient, the initial discussion on mailing lists[2] indicates that some use cases and features are not apparent—this was the motivation for conducting this analysis.

III. SPOTTER ANALYZER

Data is first collected from practitioners that use SPOTTER in their daily tasks (Section III-A). To analyze the data, SPOTTER ANALYZER employs two complementary visualizations (Section III-B).

A. Data collection

We send usage data to a server every twenty minutes in a *bundle* that is identified with a *computer UUID*. We also keep computer UUIDs of the SPOTTER developers; it lets us observe behavior differences between developers and users. By *developers* we refer to people who develop the SPOTTER tool; *users* are those who only use SPOTTER. Some developers and users contribute to PHARO IDE.

For the analysis we use the following terminology: *computers* is the collection of all computers from which we collect data; *computer* is the collection of all sessions belonging to one computer; *session* is a single-use of SPOTTER that contains events; *event* is one specific action that represents SPOTTER activity or user behavior. Each event has a timestamp and other particular information, *e.g.,* query, results, or selected item.

We analyze 2,023 sessions from 38 computers that we received between April 21, 2015 and May 26, 2015; 6 computers belong to SPOTTER developers.

[2]For example http://forum.world.st/spotter-scenario-td4811595.html, or http://forum.world.st/Spotter-td4817609.html

B. Visualizations

Timeline graph. Figure 2 shows when and how often users used SPOTTER. Each horizontal line represents one computer. At the top of the graph are SPOTTER *developers* followed by *users*. The most frequent users are placed at the top. This order assists us in distinguishing between *regular* and *sporadic users*.

Each rectangle denotes one or more SPOTTER sessions. We group together sessions that occurred within close time proximity to each other because while the duration of each session does not exceed a few minutes, the graph covers a timeframe of about one month—less than one pixel per minute. The width of each rectangle stands for duration and the color for *feature intensity usage*.

The color metric identifies who uses the features, how often they are used, and whether the user behavior changes during a time period. Figure 2 illustrates the intensity usage of the *dive-in* feature and it indicates less utilization by users. We discuss the results in more detail in Section IV.

When a user clicks on a rectangle, a *detailed timeline* is revealed. Figure 2 pictures one detailed timeline that consists of 36 sessions—some of them happened at the same time and are grouped together. Any detailed timeline can be presented as an activity diagram.

Activity diagram. Figure 2 also portrays an activity diagram with three sessions. Each line is composed of events that illustrate SPOTTER usage. We render action graphs in a *compact* way—one event next to each other with a constant distance—or in a *real time* way that indicates how much time users spent observing a particular result.

Table I gives an overview of the used symbols; next we explain some in more details. The *Query* symbol (h) is composed of rectangles—each rectangle represents one word and its width reflects word size; if a word represents a category filter, the height is smaller. In the case of Figure 1, the word is `read` and is represented as a 4-pixel wide box.

We use 13 symbols for an item *selection*: the default one (j) denotes an auto-selection that happens when SPOTTER shows the first results while searching for others—in the current implementation we use the same symbol when a user jumps between categories because we cannot properly detect this user action. Symbols on lines (k), (l), (m) relate that a user selects an item by pressing up-arrow key, down-arrow key, or by using a mouse. The first symbol on those lines denotes that an item is selected for the first time in a particular session. The second symbol explains that an item was already selected before in the same session. The last two symbols—apart from the selection repetition—indicate that a user moves to a different category.

Dive-in and *Dive-out* symbols have the shape of a stair and the deeper a user dives-in, the lower we draw the other symbols; see Figure 3f,g,h. This layout facilitates behavior observation and we can easily follow user actions and identify some common patterns.

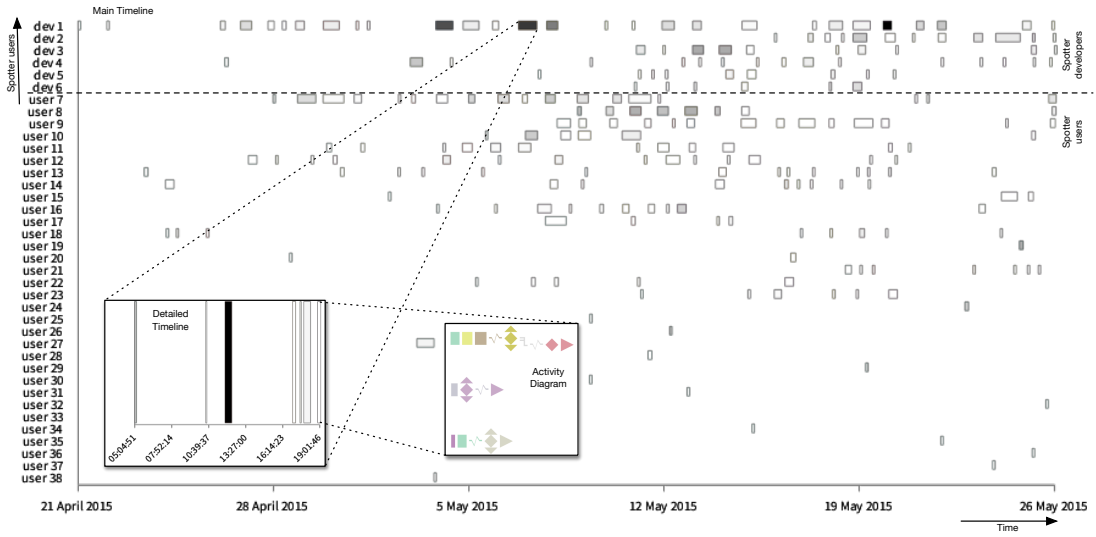

Fig. 2. Timeline graph showing SPOTTER usage for about a month. At the top are the concentrated SPOTTER developers and the most frequent SPOTTER users. Each rectangle illustrates one or more sessions that happened in a particular time period. The rectangle width relates to the time duration. The rectangle color is used for various metrics; here the blacker a rectangle is, the more often the dive-in feature is used. Any detailed timeline can be displayed as the activity diagram, each line stands for one session and color personifies the same selected item or query.

IV. RESULTS

A. Session classification

Based on the two visualizations, we identify five session types: empty, direct, first level dive-in, complex, and delayed exit. Table II characterizes the overall distribution of the classification in terms of mean, minimal, and maximum spent time.

Empty session. Figure 3a,b presents two typical *empty sessions*: in the first case a user opens and immediately closes SPOTTER; in the second case a user pastes or writes a query but also instantly closes the tool. We observe 5% (106 out of 2,023) of empty sessions with an average time of 3 seconds. In these cases we assume that users at first considered using SPOTTER, but immediately changed their minds.

Direct session. The *direct session* does not contain a dive-in action; we notice 80% (1,609 out of 2,023) of direct sessions with an average time of 9 seconds. Figure 3c,d,e presents three direct sessions: first, a user pastes a query and browses the result. In this particular example the user pastes the URL of a published script that is afterwards displayed. In the second example, as a user progressively writes a query, SPOTTER places the initial result in the first position, and the user presses enter to browse it. In both cases we suppose that users know exactly what they need and SPOTTER is used to reach that data.

The third example illustrates a user searching for a method. During the first part called "unsupported feature" the user writes a query consisting of two words and expects that SPOTTER finds items that contain both words. However this is currently not a supported feature and the tool does not present any result. We discuss it in detail in Section IV-B. The user deletes the original query, writes a new one, and observes results with 1 class and with 5 out of 28 methods. We suspect two reasons why the user does not browse any method: the user is not satisfied with the results or only needs to know a method name and does not need to browse it in another tool.

One level dive-in session. The *One level dive-in session* is a session where a user dives-in only one level deep; we collect 10% (196 out of 2,023) of this sessions with average time 52 seconds. The dive-in action can be triggered several times during one session. Figure 3f,g presents two consequences of the aforementioned missing feature that we examine in detail in the *multiple-word query* paragraph.

Complex session. The *complex session* is a session where a user dives-in more then one level. Figure 3h outlines an example. As it only represents 3% (66 out of 2,023) of the collected sessions, we do not review them in this paper.

Delayed session. The *delayed session* is a session where the time gap between the last and the next-to-last event lasts more than 3 minutes; we detect 2% (46 out of 2,023) of delayed sessions. There are 21 out of 46 sessions with the gap between 10 minutes to 13 hours. It is not apparent why it happens and it is subject of the future investigation if it is a real user behavior or a bug.

B. Answering RQ1 and RQ2

RQ1. Missing features. We detect multiple-word queries as the important missing feature.

Multiple-word query. We observe that 25% (8 out of 32) of users—and even one SPOTTER developer—used multiple-word queries. Figure 3e demonstrates such a session; a user writes a query of two words, but there is no outcome. In total we detect 33 occurrences in 17 sessions. We did not expect

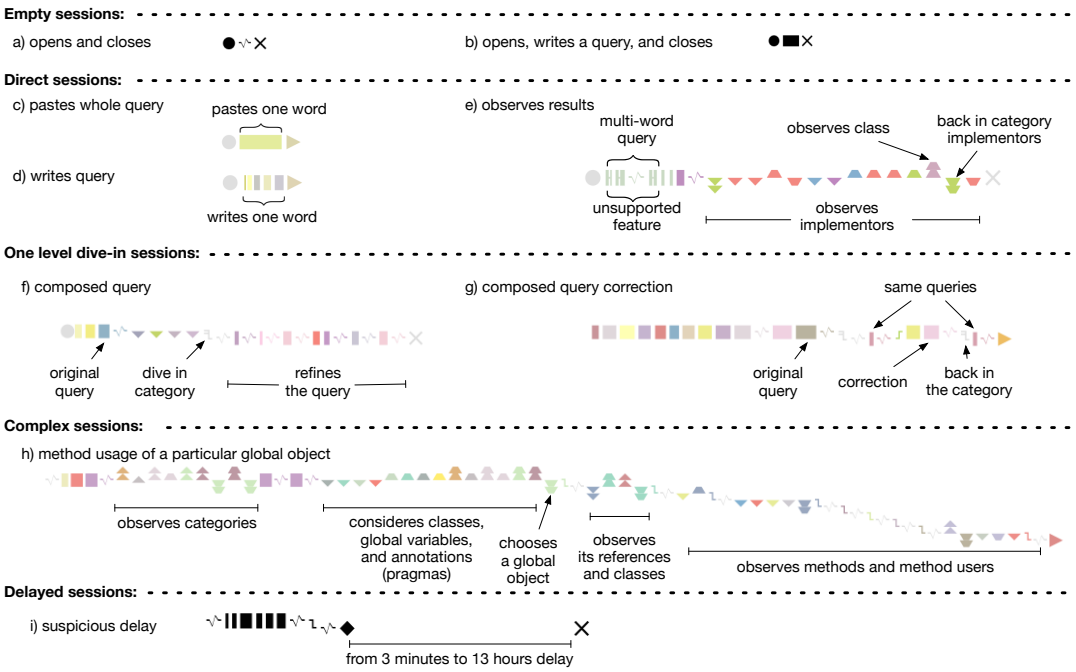

Fig. 3. Session examples. We identify five session types and set out examples for each one. Every color is associated with a particular selected item or query.

TABLE I
RELATION BETWEEN COLLECTED EVENTS AND SYMBOLS USED IN THE ACTIVITY DIAGRAM.

Event	Symbol	Description
(a) Act-On, (b) Exit	▶ ✕	Trigger default action (browse) on a selected item or close SPOTTER.
(c) Show or (d) Hide Preview	● ○	Display or hide the SPOTTER preview.
(e) Dive-In Selection or (f) Category, (g) Dive-Out	↴ ↰ ↱	Traverse items.
(h) Query	▮ ▮▮ ▮	One word query; two words query; one word query with a category filter.
(i) Search Finished	∿	SPOTTER search finished and all results are available.
(j) Selection	◇	Auto-selection of an item, or a selection without detailed information.
(k) Keyboard Selection by Up-Arrow Key	▲ ▲ ⏫ ⏫	The first time or repeatedly selected item in the same session;...
(l) Keyboard Selection by Down-Arrow Key	▼ ▼ ⏬ ⏬	...the first time or repeatedly selected item in different category then before.
(m) Mouse Selection	◆ ● ⏫ ⏬	The same information when a user interacts with a mouse.

TABLE II
SESSION CLASSIFICATION.

Classification Name	Quantity Count	Quantity [%]	Mean [hours:minutes:seconds]	Min [hours:minutes:seconds]	Max [hours:minutes:seconds]
Empty	106	5	0:00:03	0:00:00	0:00:19
Direct	1,609	80	0:00:09	0:00:01	0:22:52
One Level Dive-In	196	10	0:00:52	0:00:04	0:58:53
Complex	66	3	0:01:07	0:00:07	0:06:24
Delayed Exit	46	2	1:28:31	0:01:19	13:35:22
Total	2,023	100			

a lot of such incidents because users learn that they cannot write the multiple-word queries and they adapt to the tool.

Figure 3f displays a workaround. Let us say that a user searches for a method that contains the words `read` and `stream`. The user first writes the word `read`, then dives-in the *methods* category, and refines the result by writing `stream`. The workaround has two consequences: a) the user has to decide in advance which category is worth exploring because it is not possible to search for all items in any category that contains words `read` and `stream`; b) if the user needs to change the word `read` with the word `write`, it is necessary to dive-out, rewrite the first query, dive-in category, and write again the word `stream`—Figure 3g relates it.

RQ1 summary. We conclude that multiple-word queries is an important missing feature that should be taken into account for the next SPOTTER version.

RQ2. Unused features. The following paragraphs present four features that are seldom used: category filter, dive-in, searched information, and custom extensions.

Category filter. 19% (6 out of 32) of users used the category filter. Those 6 users used it in 3% (16 out of 522) of the sessions and in 3% (63 out of 2,002) of the queries. On the contrary, 67% (4 out of 6) of developers used it in 11% (63 out of 587) of the sessions and in 13% (300 out of 2,316) of the queries.

We observe that the category filter feature is largely ignored;

81% of users do not use it. We suppose the main reason is that the feature is not exposed in the UI and users are not informed about it.

Dive-in. We notice that 56% (18 out of 32) of users used the dive-in feature at least once and we can assume that they are aware of this feature; all SPOTTER developers used dive-in. Intensity usage metric reveals that only 5 users employ the dive-in regularly. Developers used dive-in in 18% of sessions and users used it in 11% of sessions. It indicates that users are aware of this feature, but they are likely not familiar with its usage.

Searched information. Users most often choose items in the following categories: *classes* (46%), *implementors* (28%, methods), *history* (10%, previously selected items), *menu* (5%, main menu in PHARO IDE), packages (3%), senders (2%), and in 51 other categories (6%)—in summary 84% of the total information is searched in three categories.

Before introducing SPOTTER, the common PHARO IDE search tools covered classes, implementors, senders, or pragmas (method annotations); SPOTTER exposes information that is not apparent and we should inform users about it, *e.g.,* examples, code critics, clipboard history, cached and named scripts, or help topics.

Custom extension. With the current limitation, we note that only one user extends SPOTTER by searching for items in a category named "Mongo databases." We expect that users need to find out use cases and learn about the extension mechanism; recently a user wrote that he prefers a different search flow when looking for methods[3]. It indicates that users are not familiar with the extensibility feature.

RQ2 summary. We should emphasize the use of features that are typically ignored. One option is to propose tutorials for customizing SPOTTER as well as tips based on the activity of the user, *e.g.,* informing a user who does not use dive-in that the feature exists. Another option is to rethink the contemporary UI by exposing the category filter or introducing multiple-word queries.

V. RELATED WORK

Murphy *et al.* introduce *Mylyn Monitor* [2] for Eclipse with the intention of understanding what features and plug-ins developers use. Based on 41 Java software developers, they report about the utilization of Eclipse views, plug-ins, commands, and refactoring tools.

Vakilian *et al.* argue that current existing data sources are inadequate for answering questions such as why automated refactoring are seldom used, and propose CODINGSPECTATOR [3], a tool for collecting the usage data of Eclipse refactoring tools. The collected data indicates that developers prefer small changes using refactoring tools and rarely check the refactoring preview window [4].

Yoon *et al.* developed an Eclipse plug-in called AZURITE [5] that aims to answer common questions about the code change history. They present two visualizations: a timeline visualization and a code history diff view, that both actively interact with the Eclipse code editor.

DFLOW presents a visual analysis of development sessions from the UI perspective [6]; the authors collect UI-level events and analyze developer behavior—*e.g.,* navigating, editing, or understanding code—and identify developer strategies when dealing with a change task.

Our work is different from the related work in two perspectives: first, the related work collects general user interaction data—*e.g.,* mouse clicks, keyboard usage—while we collect specific commands triggered on SPOTTER; second, the related work characterizes rather broader behavior—*e.g.,* navigating or editing code—we developed a visualization that shows user activities and we identify missing and unused features.

VI. CONCLUSION

The paper presents SPOTTER ANALYZER, a visual tool that analyzes how users use SPOTTER as well as what features are missing or rarely used. We identify that the multiple-word query is a substantial absent feature which may also be the general cause for misunderstanding the usage of SPOTTER.

The data emphasizes that the strong features of SPOTTER—like custom extensions or a large amount of scanned information—are not exploited and we should focus on it in two possible directions: first, encourage users to find out new use cases that better fit their needs; second, reconsider the current UI and the way features are exposed.

The strong benefit of the SPOTTER ANALYZER is that once we change SPOTTER or promote it, we can observe an impact comparing the historic data using SPOTTER ANALYZER—for the future work, we will schedule the SPOTTER improvements, tips and tutorials promotions and analyze the consequences.

Acknowledgment. Juraj Kubelka is supported by a Ph.D. scholarship from CONICYT, Chile. CONICYT-PCHA/Doctorado Nacional/2013-63130188. Andrei Chiş is supported by the Swiss National Science Foundation through the project "Agile Software Assessment" (SNSF project Nr. 200020-144126/1, Jan 1, 2013 - Dec. 30, 2015). Romain Robbes is partially funded by FONDECYT project 1151195. We also thank Renato Cerro for his feedback.

[3] http://bit.ly/1HPVhIq

REFERENCES

[1] E. R. Murphy-Hill, C. Parnin, and A. P. Black, "How We Refactor, and How We Know It," *IEEE Trans. Software Eng.*, vol. 38, no. 1, pp. 5–18, 2012.

[2] G. Murphy, M. Kersten, and L. Findlater, "How are Java software developers using the Elipse IDE?" *Software, IEEE*, vol. 23, no. 4, pp. 76–83, 2006.

[3] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, R. Z. Moghaddam, and R. E. Johnson, "The need for richer refactoring usage data," in *Proceedings of the 3rd ACM SIGPLAN workshop PLATEAU 2011*, Eds. ACM, 2011, pp. 31–38.

[4] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson, "Use, disuse, and misuse of automated refactorings," in *ICSE 2012*, pp. 233–243.

[5] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of fine-grained code change history," in *2013 IEEE Symposium on Visual Languages and Human Centric Computing, 2013* IEEE, 2013, pp. 119–126.

[6] R. Minelli, A. Mocci, M. Lanza, and L. Baracchi, "Visualizing Developer Interactions," in *Second IEEE Working Conference on Software Visualization, VISSOFT 2014*, 2014, pp. 147–156.