Combining Eye Tracking with Navigation Paths for Identification of Cross-Language Code Dependencies

Martin Konopka
Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
martin_konopka@stuba.sk

ABSTRACT

In recent years, fine-grained monitoring of software developers during software development and maintenance activities has increased in popularity, together with use of devices for eye tracking and recording developer's biometric data. We look for everyday application of such data to support developers in their work. In this paper we discuss an approach to identify potential code dependencies in source code, even when written in different programming languages, by combining identification of areas-of-interest in source code using eye tracking with developer's navigation paths. Our plan is to evaluate it with data of developers working on real development tasks.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *documentation, restructuring, reverse engineering and reengineering.*

General Terms

Experimentation, Human Factors, Measurement.

Keywords

Eye Tracking, Interaction Data, Potential Code Dependencies.

1. INTRODUCTION

For the long time, the prevalent sources of information about source code were based on static and dynamic analysis of its contents [8], later supplemented by monitoring a developer in revision control and task management systems, followed by interaction data from development tools [15]. External devices, like eye tracking, webcam, or pressure sensors, further extend possibilities of monitoring a software developer during software development and maintenance activities [6]. The first of the two common use case scenarios for gathering such data is to evaluate a developer herself, e.g., for ability to comprehend code [1, 9], or her expertise [4, 7]. Here, we focus more on the second use case, and thus on identifying information about source code that is initially hidden but may be revealed by means of developer's activity [10, 11, 15], as an implicit feedback to source code.

In our work we use eye tracking data and developer's interactions in an integrated development environment (IDE) for identifying source code entities and their connections [8, 12, 13] without analyzing the code (we use areas-of-interest). Such method may be used to uncover dependencies between entities in different programming languages, e.g., a client JavaScript application referencing a REST web service in C# [16].

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2807561

2. RELATED WORK

Traditional approaches for identifying code dependencies are based on syntactic analysis [8]. However, several authors have used interaction data from an IDE [15], e.g., Mylyn [10], or PerConIK [3], for identifying connections between source code entities, e.g., as interaction couplings [21], traceability links [18, 20], potential dependencies [11, 12], or recommending next navigation steps [5]. All these works differ in details of used interactions and their processing. In addition to other use cases for interaction data, e.g., assisting comprehension [3, 17, 18], or maintaining mental models and task contexts [7, 10, 14]; identifying and providing source code dependencies when syntactic analysis is not possible [11, 12, 16] is different approach to answer this problem than using dynamic analysis of source code [16, 19].

Unfortunately, interaction data still lack information whether a developer reads code when not interacting with it at all [15]. This inspires authors to employ eye tracking [9, 17, 18] for revealing developer's behavior and interest in specific blocks of code [6], e.g., how she comprehends code [1, 9, 18], debugs code [2, 9], or even to understand her expertise [4].

3. DEVELOPER'S ACTIVITY AND GAZE

The most common interaction that developers perform within development activities is navigation in a space of source code entities, i.e., source code documents, types, or their members. We may distinguish between these ways to navigate in an IDE:

- Choosing entities in a structural view on source code, e.g., project or package explorer, class view, search results, etc.
- Switching tabs of recently opened documents in a code editor, e.g., sequentially, or directly choosing a document.
- Changing viewport of an actual source code document using mouse or keyboard, e.g., scrolling in a code editor.
- Jumping between types and their members in source code using references – requires source code analysis.

Developers arbitrarily navigate in source code, based on their current task and activity [10, 15], and then perform other interactions, e.g., comprehend code, solve a problem while debugging, or maintain client and server code successively.

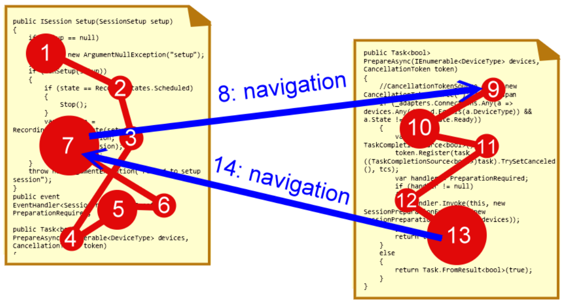

Figure 1. Example of developer's gaze in two source code documents and navigation between them.

Based on that, we expect the source and target entities of a navigation to be somehow connected, being it syntactically, or even semantically [11].

Eye tracking technology, e.g., by Tobii (http://www.tobii.com/), allows us to record developer's areas-of-interest in a source code document [18]. In comparison to recording information just about source and target documents of a navigation, recording developer's fixations on lines of code (or blocks) tells us the exact parts of those documents that she is interested in. Figure 1 shows an example scenario of a developer exploring one source code document, interacting with its fragment, then switching to another one, finding a related fragment, and then going back.

We can get similar, but less-detailed, information about developer's gaze from the current viewport of a source code document [5]. However, eye tracking metrics may be later used for detailed evaluation of developer's gaze, e.g., pupil size [6].

4. IDENTIFICATION OF DEPENDENCIES

Monitoring developer's navigation paths and eye tracking provides us with developer's implicit feedback on source code. We aim to employ it to infer existence of potential source code dependencies [12], e.g., method calls or type references.

As we shown in [11], static dependencies may be inferred from developer's interactions. However, we were able to identify dependencies between source code documents only, not entities contained in them. We expect that before an attempt to navigate, developer focuses on a code element related to an element which she works with right after the navigation. Using developer's navigations to traverse a source code space, we connect fixations between documents. Attempted navigation results in changing the displayed content in a code editor, and, although a developer has done even no saccades, she then fixates her gaze on different content than before. Finally, we expect those specific places in code to be potentially dependent.

To make our method independent from any syntactic analysis of underlying source code contents, we use these IDE interactions:

– Navigation between documents – open, switch to, or close.

– Change in viewport of a document – scrolling.

We use actual contents of a document only to infer areas-of-interest in source code that we represent by the starting and ending lines in a document. We propose to assign a sequence of the last *N* fixations in a source document, and the first *M* fixations in a target document (e.g., *N=M=10*), for each performed document navigation, together with duration of fixations, dwell time in the target document [11], and other eye tracking metrics. Then we aggregate fixations (coordinates on the screen transformed into line and column positions in source code documents) into area(s)-of-interest, i.e., source code elements (possibly blocks of code) as the source and target places of performed navigation. With this approach we identify potential dependencies between not just documents, but more fine-grained source code entities within and between documents [11, 12]. In the example shown in Figure 1, the areas underneath fixations 7 and 13 may be potentially dependent.

5. EVALUATION PROPOSALS

To experimentally evaluate proposed approach we collect recordings of various development sessions in two stages.

In the first stage of experimental evaluation of proposed approach, we plan to use Tobii Studio with Tobii X60 and TX300 devices for controlled experiment of a developer studying source code known to him and editing it. After that, we will manually analyze and annotate data to explore how the nearest eye fixations in source code documents before and after a navigation correlate with actual static dependencies in source code. We expect the navigation interaction itself to make noise in tracking developer's gaze in documents, e.g., checking the list of documents when switching tabs, or navigating to wrong place.

In the second stage, we equip developers with Tobii X60 or EyeX devices, together with our tools to record data from them, as well as to record screencast, and interactions in Microsoft Visual Studio or Eclipse IDEs [3]. This setup may be more feasible for developers to track their own software development activities, rather than being set up in an experiment room [1]. We expect to gather noisy data of real development scenarios, although unknown to us, and not repeatable among participants. Optional audio (think aloud) recording may help us to overcome this problem, as well as asking developers to textually describe their development sessions. To avoid developers' misconceptions on privacy leaks, they will be selected specifically for the experiment, and will be able to turn off the recording anytime.

6. DISCUSSION AND CONTRIBUTION

In this work, we use eye tracking data and navigation paths for identification of potential source code dependencies, independently from programming languages used. Although navigation paths may be taken even at random, in many cases, they implicitly reveal static dependencies in source code [11]. As a result, we identify potential dependencies even between entities in different languages, e.g., JavaScript and C#, C++, Java, etc.

We chose to use eye tracking for our work because it provides finer data than just recording viewports in code editor [5], and we may apply eye tracking metrics for weighting and validating identified potential dependencies [11].

For experimental evaluation we record eye tracking and interaction data of both controlled and uncontrolled development sessions using Tobii devices and tools provided by the PerConIK project (http://perconik.fiit.stuba.sk/) [3]. We see usage of such data also for other studies, thus we plan to release them to public.

7. ACKNOWLEDGMENTS

This work was partially supported by the Scientific Grant Agency of Slovakia, grant No. VG 1/0752/14; and it is the partial result of the Research & Development Operational Programme for the project "University Science Park of STU Bratislava", ITMS 26240220084, and the project PerConIK, ITMS 26240220039, both co-funded by the ERDF.

8. REFERENCES

[1] Bednarik, R., Tukiainen, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proc. of the 2006 Symposium on Eye Tracking Research & Applications.* ETRA '06. ACM, 125-132.

[2] Bednarik, R., Tukiainen, M. 2008. Temporal eye-tracking data: evolution of debugging strategies with multiple representations. In *Proc. of the 2008 Symposium on Eye Tracking Research & Applications.* ETRA 2008. ACM, 99-102.

[3] Bielikova, M., Polasek, I., Barla, M., et al. 2014. Platform independent software development monitoring: design of an architecture. In *Proc. of the 40th International Conference on Current Trends in Theory and Practice of Computer Science.* SOFSEM 2014. Springer-Verlag, 126-137.

[4] Busjahn, T., Schulte, C., Sharif, B., et al. 2014. Eye tracking in computing education. In *Proc. of the 10th Annual Conference on International Computing Education Research.* ICER '14. ACM, 3-10.

[5] DeLine, R., Khella, A., Czerwinski, M. et al. 2005. Towards understanding programs through wear-based filtering. In *Proc. of the 2005 ACM Symposium on Software Visualization.* SoftVis '05. ACM, 183-192.

[6] Fritz, T., Begel, A., Müller, S. C., et al. 2014. Using psycho-physiological measures to assess task difficulty in software development. In *Proc. of the 36th International Conference on Software Engineering.* ICSE 2014. IEEE CS Press, 402-413.

[7] Fritz, T., Murphy, G.C., Murphy-Hill, E., et al. 2014. Degree-of-knowledge: Modeling a developer's knowledge of code. In *ACM Transactions on Software Engineering and Methodology*, Vol. 23 Issue 2, Article 14. TOSEM. ACM.

[8] Grove, D., Chambers, C. 2001. A framework for call graph construction algorithms. In *ACM Transactions on Programming Languages and Systems,* Vol. 23 Issue 6. TOPLAS. ACM, 685-746.

[9] Hejmady, P., Narayanan, N. H. 2012. Visual attention patterns during program debugging with an IDE. In *Proc. of the Symposium on Eye Tracking Research and Applications.* ETRA '12. ACM, 197-200.

[10] Kersten, M., Murphy, G.C. 2006. Using task context to improve programmer productivity. In *Proc. of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* SIGSOFT '06/FSE-14. ACM, 1-11.

[11] Konopka, M., Bielikova, M. 2015. Software developer activity as a source for identifying hidden source code dependencies. In *Proc. of the 41st International Conference on Current Trends in Theory and Practice of Computer Science.* SOFSEM 2015. Springer-Verlag, 449-462.

[12] Konopka, M., Navrat, P., Bielikova, M. 2015. Poster: Discovering code dependencies by harnessing developer's activity. In *Proc. of the 37th International Conference on Software Engineering.* ICSE 2015. IEEE CS Press, 801-802.

[13] Krämer, J.-P., Karrer, T., Kurz, J., et al. 2013. How tools in IDEs shape developers' navigation behavior. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. ACM, 3073-3081.

[14] LaToza, T.D., Venolia, G., DeLine, R. 2006. Maintaining mental models. In *Proc. of the 28th International Conference on Software Engineering.* ICSE '06. ACM, 492-501.

[15] Maalej, W., Fritz, T., Robbes, R. 2014. Collecting and processing interaction data for recommendation systems. *Recommendation Systems in Software Engineering*. Ch. 7, Springer-Verlag, 173-197.

[16] Nguyen, H. V., Kästner, C., Nguyen, T. N. 2014. Building call graphs for embedded client-side code in dynamic web applications. In *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. ACM, 518-529.

[17] Rodeghero, P., McMillan, C., McBurney, P. W., et al. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *Proc. of the 36th International Conference on Software Engineering*. ICSE 2014. IEEE CS Press, 390-401.

[18] Sharif, B., Kagdi, H. 2011. On the use of eye tracking in software traceability. In *Proc. of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering.* TEFSE '11. ACM, 67-70.

[19] Spasojević, B., Lungu, M., Nierstrasz, O. 2014. Mining the ecosystem to improve type inference for dynamically typed languages. In *Proc. of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software.* Onward! '14. ACM, 133-142.

[20] Walters, B., Shaffer, T., Sharif, B., et al. 2014. Capturing software traceability links from developers' eye gazes. In *Proc. of the 22nd International Conference on Program Comprehension*. ICPC 2014. ACM, 201-204.

[21] Zou, L., Godfrey, M. W., Hassan, A. E. 2007. Detecting interaction coupling from task interaction histories. In *Proc. of the 15th International Conference on Program Comprehension.* ICPC 2007. IEEE CS Press, 135-144.