Developer Interaction Traces backed by IDE Screen Recordings from Think aloud Sessions

Aiko Yamashita
Oslo Metropolitan University
aiko.fallas@gmail.com

Fabio Petrillo
Concordia University
fabio@petrillo.com

Foutse Khomh
Polytechnique Montréal
foutse.khomh@polymtl.ca

Yann-Gaël Guéhéneuc
Concordia University
yann-gael.gueheneuc@concordia.ca

ABSTRACT

There are two well-known difficulties to test and interpret methodologies for mining developer interaction traces: first, the lack of enough large datasets needed by mining or machine learning approaches to provide reliable results; and second, the lack of "ground truth" or empirical evidence that can be used to triangulate the results, or to verify their accuracy and correctness. Moreover, relying solely on interaction traces limits our ability to take into account contextual factors that can affect the applicability of mining techniques in other contexts, as well hinders our ability to fully understand the mechanics behind observed phenomena. The data presented in this paper attempts to alleviate these challenges by providing 600+ hours of developer interaction traces, from which 26+ hours are backed with video recordings of the IDE screen and developer's comments. This data set is relevant to researchers interested in investigating program comprehension, and those who are developing techniques for interaction traces analysis and mining.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; Software development process management;

KEYWORDS

empirical study, industrial data, interaction traces, log mining, program comprehension, programming flow

ACM Reference Format:
Aiko Yamashita, Fabio Petrillo, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2018. Developer Interaction Traces backed by IDE Screen Recordings from Think aloud Sessions. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196457

1 INTRODUCTION

There are two well-known difficulties to test and interpret methodologies for mining developer interaction traces: first, attaining enough large datasets so that mining or machine learning (ML) techniques could provide reliable results; and second, the lack of "ground truth" or empirical evidence that can be used for triangulation, or for verifying the accuracy and correctness of analysis outcomes. Moreover, relying solely on interaction traces limits our ability to take into account contextual factors that could affect the applicability of the proposed techniques in other contexts. One final big caveat of lacking contextual data is that it stops us from fully understanding the mechanics behind an observed phenomenon.

Context can impact the outcome of a software engineering activity [11] and by analysing it, we can attain meaningful interpretations to our measurements and observations. Contextual factors such as programming skill, the problem at hand, industrial domain, are such factors [4, 6], but they are not always available nor are measured systematically. Also, due to limitations on the study design, or resources available to perform the study, *triangulation*[1] is more often the exception rather than the rule. With an increased presence of Artificial Intelligence, Machine Learning and Data Science techniques in the mainstream industrial arena, the usage of concepts such as *Contextual Intelligence*[2] has gained momentum, and is currently perceived by data intelligence companies as a promising arena to attain more robust and adequate insights/results from big data analytics [7]. Although contextual analytics can be traced back to Management and Business Analytics [14], it has permeated other domains such as health care [2], and web analytics[8].

Ironically, datasets that are rich in volume as well as contextual data or meta-data are scarce, whiles they seem to be critical not only for advancing in the validity and robustness of scientific research, but also for attaining Linked Open Data [21].

The data presented in this paper intends to alleviate these challenges by providing 600+ hours of developer interaction traces, from which 26+ hours are backed by video recordings of the IDE screen and developer's comments. The videos were anonymised and in order to keep the programmers' identities concealed, and were collected during a study involving 6 software developers who

[1]In the social sciences, triangulation is often used to indicate that more than two methods are used in a study, with a view to double (or triple) checking results. This is also called "cross examination."

[2]Defined by Mayo & Nohria [10] as: "*the ability to understand the limits of our knowledge and to adapt that knowledge to an environment different from the one in which it was developed.*"

were followed for a period of 4 months. This data set is relevant to researchers interested on investigating program comprehension, and those who are developing techniques for interaction traces analysis/mining for different purposes.

The remainder of this paper is organized as follows: Section 2 provides a brief background of the study from which the data was obtained. Section 3 describes the data being released. Section 4 explains how the data can be accessed and used as well as caveats and limitations. Section 5 concludes and presents future work.

2 BACKGROUND OF THE DATA SET

This data set is derived from an industrial case study conducted by Simula Research Laboratory in 2008, with the objective of investigating the effect of code smells on the evolution and quality of real-life, industrial software systems. Four java-based web systems were involved in the study. Their main functionality consisted of keeping track of the empirical studies conducted at a research center, including information such as the responsible for the study, participants, data collected, publications from the study, etc.

Six software professionals were recruited from a pool of 65 participants of a previous study on programming skill [5] who displayed similar programming abilities. The developers were asked to perform three (identical) maintenance tasks, two adaptive and one perfective on two out of the four systems analysed. This assignment resulted in 3 projects per system, i.e., 6 developers x 2 systems = 12 projects (cases) in total. The development took place entirely at the developers' company sites and the first author of this paper was present in the sites for the entire duration of the project. The developers were given no information on what the study entailed. Eclipse was used as the development tool, together with MySQL (www.mysql.com) and Apache Tomcat (http://tomcat.apache.org). Subversion or SVN (http://subversion.apache.org/) was used as the versioning system. The work reported in [15] provides a complete account of the context and methodology of the study.

3 DATA DESCRIPTION AND RELEASE APPROACH

3.1 Data Description and Collection

The dataset comprises of two parts: 1) Videos from the think-aloud sessions, and 2) Developer Interaction Traces (Logs). During the study, screen-recorded think-aloud sessions (30-40 minutes) were conducted every another day to observe the developers in their daily activities. ZD Soft Screen Recorder[3] was used to capture the screen during the sessions. The interaction traces were captured via an Eclipse plug-in called Mimec [9]. Mimec can capture Eclipse IDE events, such as editing source files, scrolling the source code window, switching between open files, expanding/collapsing trees in the package explorer, selecting Java elements (classes, methods, and variables), and running Eclipse "commands" (e.g., copy, save, and go to end of line). The interaction traces were stored as Comma-Separated Value (CSV) files, where every line corresponds to an event, a single observation generated by Mimec. Each event consists of six pieces of data, as depicted in Table 1. A Java program was written to identify the elapsed time between the different activities

[3]https://www.zdsoft.com/screen-recorder

Table 1: Description of data contained in an event.

1.	*Timestamp*	*Time* in milliseconds when the event was recorded
2.	*Date*	*Time* the event was observed by Mimec (similar to #1)
3.	*Kind*	*Kind* of event: *edit*, *selection*, *command* or *preference*
4.	*Target*	A Java element (if any) that is the subject of the interaction, such as the name of the file selected, or the name of the class/method being edited.
5.	*Origin*	The part of Eclipse that generates the interaction (e.g., Package Explorer, Editor)
6.	*Delta*	An attribute (if any) containing relevant meta-information.

by truncating the consecutive events annotated with the same kind of event, target, and origin. An additional heuristic was used to handle two particular situations. In the project, the developers had to work with multiple environments besides the Eclipse IDE. They had to: 1) Look at documentations and 2) Run the application and interact with the GUI (website) component of the systems, and 3) Work on the DB via tools other than Eclipse. For all these events, the developer would leave the IDE. Mimec does not record when the developer leaves the IDE but does so only when they return to the IDE. This situation would yield inaccurate results in some cases. For example, a developer may first select a file and then leave the IDE to take a coffee break, and all the time spent on the coffee break will be assigned to the action "select file." Another problem of Mimec is that for certain editing commands (e.g., copy, paste, and cut), it does not register the file in which the activity was done. To solve the "idle time" problem, a lookup table was created with average times of all types of activities from all the logs of all developers. The average values excluded any log entries that occurred just before the activity "*Go back to IDE*" (because those are precisely the ones that we want to correct). Sample sizes used to compute those averages were very high, and standard deviations were very low, so they were considered trustworthy. The algorithm for calculating the time was adjusted, so when any activity was followed by "*Go back to IDE*", then the algorithm will evaluate:

Case 1: If the elapsed time between entries is equal or lower than the average time indicated in the lookup table, assign the whole elapsed time to the entry. and

Case 2: If the elapsed time between entries is higher than the average time indicated in the lookup table, assign the average time from the lookup table to the entry and the elapsed time minus average time to "Unknown activities outside IDE."

To solve the "missing file" problem, the filename contained in the closest preceding entry to any log entry containing any of the problematic commands (copy, paste, and cut) was used as the filename. The source code written to process the interaction traces is available here: http://doi.org/10.5281/zenodo.1248461

3.2 Process for Releasing the Data

Prior releasing the video data from the think aloud sessions, we anonymised the voices of the participants and blurred the areas in the screen that made any reference to the name of the company or employees of the company hired to perform the maintenance tasks.

Step 1: Audio anonymisation – To anonymise an audio file, we changed the participant's voice without a disagreeable distortion while reducing the possibility of reversing the change to obtain the original audio. We increased or decreased the voice frequency of the participants to achieve a pitch similar to that of a young male voice. We used *Audacity*[4] to modify the audio tracks of videos.

Step 2: Video anonymisation – We used Filmora[5] to anonymise sensitive information in the videos. Filmora has a feature that allows us to select a region on the video and apply the blurring effect. The analysis of the video had to be done frame by frame because the sensitive data could appear everywhere and very quickly in videos. This blur action can be done on the same specific region from the beginning of the video to the end, or it can be applied on specific frames in the video, using cutting tools.

Step 3: Rendering – This step consists of combining the Video, Audio and the effect, generating a final video output. We adopted as a standard use, the original resolution of the video with MP4 as output.

Step 5: Quality Verification – The anonymised videos were examined in its entirety by an independent reviewer, and corrections were made afterwards. A final random subsample was selected for a final check by all the authors of the paper.

Step 6: Meta-data (system time) extraction – A consequence of video anonymisation (in specific, the blurring of the bottom bar of the window) is that the System time was concealed. In order to enable a direct mapping between the videos and timestamps in the interaction traces, we extracted that information from the original videos and made them available as meta-data.

As for the interaction traces, there was no need for anonymisation, given that the files do not contain any personal data on the developers nor the company in which they worked. The filenames of the systems (and their original developers, which did not participate in this study) are publicly available, given that the systems were released under Creative Commons Attribution 4.0 (via the MSR 2017 Data Showcase paper [17]) by Simula Research Laboratory, who owed the source code and conducted the study.

4 DATA USAGE

4.1 How to access the data?

The structure of the dataset is described by Fig. 1. At high level, the data is organized according to the developer. Under each Developer{1–6} folder, there is a folder containing the user interaction traces and the videos from the think aloud sessions. The Interaction Traces folder contain all three versions of the logs: the 1-Original or raw logs, the 2-Annotated (where each single event was annotated with its corresponding high-level activity label) and the 3-Truncated logs (where consecutive events under the same activity label are truncated and the total elapsing time under those truncated events are calculated). The sub-folder Think aloud videos simply contain the anonymised videos. The repository is connected to the GitHub repository containing the Java code written to process the interaction traces. Finally, there is an excel sheet containing the meta-data of the think aloud videos. The dataset is available at: http://doi.org/10.5281/zenodo.883813

[4]http://www.audacityteam.org/
[5]https://filmora.wondershare.com/

4.2 How has the data been used?

The data has been used in multiple works [12, 13, 15, 16, 18–20] pertaining the study of source code properties (i.e. code smells), and their role on the maintainability and quality of Object Oriented systems.

4.3 Potential usage scenarios

Due to the nature of this dataset, we envision countless usage scenarios within program comprehension and data mining. Here are some examples:

Studies on programming behavior. This data can provide better insights for triangulating results from automated mining, with in-depth investigation of programming behavior.

Studies on the impact of different variables on programming behavior. This data has primarily been used to investigate the effect of code properties on software maintainability/quality, but other metrics and factors can be extracted from the study, and investigated by using the interaction traces and the videos.

Benchmarking of tools/methodologies. This data set and the underlying systems can be used for benchmarking purposes, when evaluating new tools for metrics detection, defect extraction, or any other methodologies.

Task/context extraction. The data allows to experiment with techniques/tools/methods that can allow identifying the exact context (e.g., task) by analysing the logs and verifying them on the video samples. Such techniques have industrial applications such as the one reported by [3].

4.4 Challenges and Limitations

No complete video coverage. The logs for the developer interaction traces account for more than 600 hours of development, but the think-aloud videos are only 28. However, the sampling was done randomly so we hope that there is a good coverage of distinctive cases during the development.

Context of the study. The external validity of any results from this data are limited to the context of the study, in this case: medium-sized, Java-based, three-layered architecture, web-based, information systems.

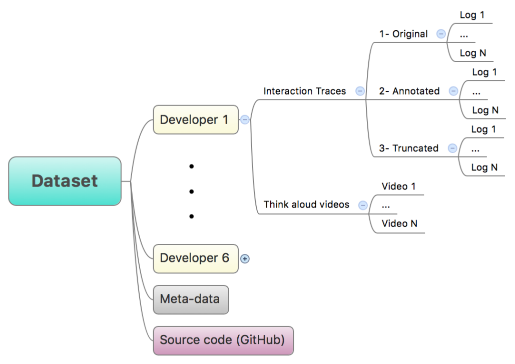

Figure 1: Organization of the data

Time frame. The data does not fully represent a long-term maintenance project, given the size of the tasks and the shorter maintenance period covered in the study. However, tasks are similar to backlog items in a single sprint/iteration.

The age of the systems. The technology used in this study is nearly 10 years (14 if the original study [1] is considered). However, there are still many industrial systems which are even older than 14 years, and the technology involved is still quite relevant to current software projects.

Tool availability. The tool used for generating the interaction traces (Mimec) is no longer available. The authors are aware of other tools such as CodingTracker, DFlow, FeedBaG, WatchDog and Flourite, which would have been more adequate than Mimec. However, the study was conducted in 2008, and Mimec was one of the few available tools at the time. Thus, interaction traces generated by alternative tools such as Flourite (https://github.com/yyoon/fluorite-eclipse) are not guaranteed to provide the same results. This also applies to the Java code written to process the interaction traces. However, there is a new tool developed at CWI, for processing interaction traces, which has yielded consistent results with this dataset (https://github.com/King07/espionage) as well as with data generated by Flourite.

Realism of the study. There will be a trade-off between the degree of realism and the degree of control in empirical studies. We believe the systems and tasks belong to a realistic setting, and special care was put in order to ensure as much as possible, a realistic project.

Hawthorne effect. Since the think aloud essentially involved observing the behavior of the developers directly, this may influence the developers' behavior during the screen capture. However, this limitation equally applies to most other qualitative studies.

5 CONCLUSION AND FUTURE WORK

We presented a data set containing 600+ hours of developer interaction traces, from which 26+ hours are backed with video recordings of the computer screen and developers' verbal comments. As such, this dataset constitutes both a large *quantitative* and rich *qualitative* (contextual) opportunity to investigate empirically programming activities, and to evaluate/validate approaches for mining interaction traces. This dataset complements primary data reported at a previous MSR (multiple *evolution histories*, and multiple *defect reports*, extracted from git and issue tracking systems). In future related work, we plan to: 1) provide concrete guidelines for sharing diverse types of empirical data from software engineering studies, and 2) release a platform to support the sharing of research data and reproducibility of studies in Software Engineering, including the incorporation of ReproZip (https://www.reprozip.org/), and Dockerfiles/Docker images.

ACKNOWLEDGEMENT

The authors thank CWI, Jurgen Vinju, the SWAT lab at Polytechnique Montréal, and Andrea Caracciolo for their invaluable help during the anonymisation of the videos.

REFERENCES

[1] Bente C D Anda. 2007. Assessing Software System Maintainability using Structural Measures and Expert Assessments. In *IEEE International Conference on Software Maintenance*. 204–213. https://doi.org/10.1109/ICSM.2007.4362633

[2] Oresti Banos, Claudia Villalonga, Jaehun Bang, Taeho Hur, Donguk Kang, Sangbeom Park, Thien Huynh-The, Vui Le-Ba, Muhammad Bilal Amin, Muhammad Asif Razzaq, Wahajat Ali Khan, Choong Seon Hong, and Sungyoung Lee. 2016. Human behavior analysis by means of multimodal context mining. *Sensors (Switzerland)* 16, 8 (2016), 1–19. https://doi.org/10.3390/s16081264

[3] Mike Barnett, Christian Bird, JoÃčo Brunet, and Shuvendu K Lahiri. 2015. Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 134–144.

[4] R. M. Belbin. 2010. *Management teams : why they succeed or fail.* Butterworth-Heinemann. 193 pages.

[5] Gunnar Rye Bergersen and Jan-Eric Gustafsson. 2011. Programming Skill, Knowledge, and Working Memory Among Professional Software Developers from an Investment Theory Perspective. *Journal of Individual Differences* 32, 4 (1 2011), 201–209. https://doi.org/10.1027/1614-0001/a000052

[6] Jean-Marie Burkhardt, Francoise Détienne, and Susan Wiedenbeck. 2002. Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. *Empirical Software Engineering* 7, 2 (2002), 115–156. https://doi.org/10.1023/A:1015297914742

[7] Tarun Khanna. 2014. Contextual intelligence. *Harvard Business Review* 92, 9 (2014), 59–+.

[8] Julia Kiseleva. 2013. Context mining and integration into predictive web analytics. *Proceedings of the 22nd International Conference on World Wide Web - WWW '13 Companion* March (2013), 383–388. https://doi.org/10.1145/2487788.2487947

[9] Lucas M Layman, Laurie A Williams, and Robert St. Amant. 2008. MimEc. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM Press, New York, New York, USA, 73–76. https://doi.org/10.1145/1370114.1370133

[10] Anthony J. Mayo and Nitin Nohria. 2005. Zeitgeist leadership. *Harvard Business Review* 83, 10 (2005).

[11] Forrest Shull, Victor Basili, Jeffrey Carver, JosÃĽ C Maldonado, Guilherme Horta Travassos, Manoel Mendonça, and Sandra Fabbri. 2002. Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*. IEEE Computer Society, Washington, DC, USA.

[12] Dag I.K. Sjoberg, Aiko Yamashita, Bente C.D. Anda, Audris Mockus, and Tore Dyba. 2013. Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE Transactions on Software Engineering* 39, 8 (8 2013), 1144–1156. https://doi.org/10.1109/TSE.2012.89

[13] Zephyrin Soh, Aiko Yamashita, Foutse Khomh, and Yann-Gael Gueheneuc. 2016. Do Code Smells Impact the Effort of Different Maintenance Programming Activities?. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 393–402.

[14] Edwin Van Bommel, David Edelman, and Kelly Ungerman. 2014. Digitizing the consumer decision journey. *McKinsey Quarterly* (2014).

[15] Aiko Yamashita. 2012. *Assessing the Capability of Code Smells to Support Software Maintainability Assessments: Empirical Inquiry and Methodological Approach.* Ph.D. Dissertation. University of Oslo.

[16] Aiko Yamashita. 2013. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. *Empirical Software Engineering* 19, 4 (3 2013), 1111–1143. https://doi.org/10.1007/s10664-013-9250-3

[17] Aiko Yamashita, S Amirhossein Abtahizadeh, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2017. Software evolution and quality data from controlled, multiple, industrial case studies. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on.* IEEE, 507–510.

[18] Aiko Yamashita and Steve Counsell. 2013. Code smells as system-level indicators of maintainability: An Empirical Study. *Journal of Systems and Software* (2013).

[19] Aiko Yamashita and Leon Moonen. 2013. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 682–691.

[20] Aiko Yamashita and Leon Moonen. 2013. To what extent can maintenance problems be predicted by code smell detection? âĂŞ An empirical study. *Information and Software Technology* 55, 12 (12 2013), 2223–2242. https://doi.org/10.1016/j.infsof.2013.08.002

[21] Anneke Zuiderwijk, Keith Jeffery, and Marijn Janssen. 2012. The necessity of metadata for linked open data and its contribution to policy analyses. In *Proceedings CeDEM2012 Conference, Donau-Universitat, Krems.* 281–294.