Noises in Interaction Traces Data and their Impact on Previous Research Studies

Zéphyrin Soh*, Thomas Drioul†, Pierre-Antoine Rappe†, Foutse Khomh*, Yann-Gaël Guéhéneuc*, and Naji Habra†
*Polytechnique Montréal, Canada; {zephyrin.soh, foutse.khomh, yann-gael.gueheneuc}@polymtl.ca
†FUNDP Namur; {thomas.drioul, pierre-antoine.rappe, naji.habra}@unamur.be

***Abstract—Context*: Developers' interaction traces (ITs) are commonly used in software engineering to understand how developers maintain and evolve software systems. Researchers make several assumptions when mining ITs, *e.g.,* edit events are considered to be change activities and the time mined from ITs is considered to be the time spent by the developers performing the maintenance task. *Goal*: We investigate the extent to which these assumptions are correct. We examine noises in developers' ITs data and the impact of these noises on previous results derived from these traces. *Approach*: We perform an experiment with 15 participants, whom we asked to perform bug-fixing activities and collect Mylyn ITs and VLC video captures. We then investigate noises between the two data sets and propose an approach to correct noises in ITs. *Results*: We find that Mylyn ITs can miss on average about 6% of the time spent performing a task and contain on average about 28% of false edit-events. We report that these noises may have led researchers to mislabel some participants' editing styles in about 34% of the cases and that the numbers of edit-events performed by developers and the times that they spent on tasks are correlated, when they were considered not to be. *Conclusion*: We show that ITs must be carefully cleaned before being used in research studies.**

Keywords—Software maintenance, interaction traces, noises, video captures, maintenance effort.

I. Introduction

Developers' activities are valuable source of information to improve their productivity [5]. Developers' interaction traces (ITs), also known as activity logs, have been used to study developers' preferred views in an IDE [10], edit styles [22], [24], and exploration strategies [19] as well as work fragmentation [16], maintenance effort [14], [18], change prediction [1], and impact analysis [23]. Developers' ITs have also been used to provide tool support through code completion [13] and recommendations of relevant program entities [4], [8], [9]. They are usually collected by monitoring tools, such as Mimec [7] or Mylyn [11].

Most of these studies depend on the accuracy of the information mined from ITs. An IT is an ordered list of events triggered by developers while performing their task. Each event has a kind (*e.g.,* selection, edit), has a start and end timestamps, and is triggered on a program entity (*e.g.,* file, class, or method). Unfortunately, ITs contain noise. Some events may (1) have a 0 duration, some events may (2) overlap with one another, and some edit events may not (3) correspond to real modifications of the source code because current tools, such as Mylyn, (1) record the 0 duration, (2) temper on-the-fly with events to ease their analyses and collect events not directly triggered by developers, and (3) generate edit events every time a developer selects a piece of text in an editor [12] even if the code does not change. Consequently, previous studies that infer developers code edit styles [22], code edit patterns [24], or recommend code entities based on these recorded edit events [5], [16], [19] are likely to have some inaccuracies.

For example, the IT in Figure 1 shows an excerpt of the edit events recorded for one developer during our empirical study (See Section III). The edit events e_1, e_2, e_3 could be labelled by Mylyn as edit-events but they could be *false* edit-events because the developer did not modify the source code (when observing the corresponding screen capture). Therefore, using this IT, the developers' edit style [22], for example, would be wrongly inferred as *edit-throughout* instead of *edit-last*, even though edit-events *really* occurred towards the end of the task.

After observing the impact that noises in developers' ITs can have on the accuracy of the analyses performed on the ITs collected by current monitoring tools, we systematically investigate these noises. Our goals are to (1) quantify noises in current ITs collected by Mylyn, (2) propose an approach to correct these noises, and (3) assess how these noises and corrections affect some previous studies. Our goals aim to help (1) researchers to mine ITs reliably and (2) tools providers to improve monitoring tools.

To reach our goals, first, we conduct a controlled experiment involving 15 developers to whom we asked to complete maintenance tasks on four open-source Java systems (ECF, jEdit, JHotDraw, and PDE). During the tasks, we collected both Mylyn ITs and video captures of developers' screens. Mylyn ITs—RITs for Raw Interaction Traces in the following—have been often used in the literature to study developers' activities. Video captures have been used as alternatives to ITs [6], [15], [20] when transcribed into ITs—VITs for Video-based Interaction Traces in the following. By comparing RITs and VITs, we observe that RITs miss on average about 6% of the time spent performing a task and contain about 28% of false edit-events. Second, we use the mean-duration of false edit-events to propose an approach to automatically correct RITs. Using our approach, we revisit some previous studies by Ying and Robillard [22] and Soh *et al.* [19] and show that these noises in ITs may have led researchers to mislabel some editing styles in about 34% of the cases. We also observe that the numbers of edit-events performed by developers and the times that they spent on tasks are not correlated.

The remainder of the paper is organized as follows. Section II summarises previous studies. Section III describes our empirical study of the noises in ITs. Section IV presents our approach to correct ITs. Section V reports our revisits of two previous studies. Section VI discusses threats to the validity of our study, approach, and revisits. Section VII concludes.

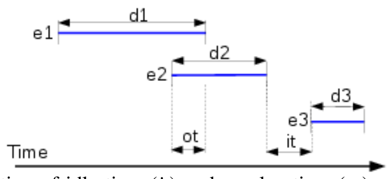


Fig. 1. Illustration of idle time (*it*) and overlap time (*ot*)

II. Related Work

Many studies have studied developers' ITs. While some previous studies focused on the analysis of ITs to understand developers' behaviour, others used ITs to study software engineering activities. Yet some others build tools to support developers in their daily work. These three uses of ITs are complementary because to provide tool support based on ITs, there is a need for a good understanding of developers' behaviour and activities.

A. Interaction Traces to Understand Developers' Behaviour

Ying and Robillard [22] studied the developers' editing style and characterised edit-first, edit-last, and edit-throughout editing styles. They observe that enhancement tasks are associated with a high fraction of edit events at the beginning of the programming session (*i.e.,* edit-first). Zhang *et al.* [24] studied how several developers concurrently edit a file and derive concurrent, parallel, extended, and interrupted file editing patterns. They stated that file editing patterns impact software quality because they are related to future bugs.

Soh *et al.* [19] classified developers' exploration strategies as referenced exploration (*i.e.,* revisitation of a set of entities) and unreferenced exploration (*i.e.,* program entities are almost equally revisited). They found that unreferenced explorations require more effort but are less time consuming than referenced explorations. Murphy *et al.* [10] analysed how developers use the Eclipse IDE, *e.g.,* which Eclipse views and perspectives they mostly use. They observed that feature usage vary but that, for example, none used the declaration view.

B. Interaction Traces to Study Software Engineering Activities

Sanchez *et al.* [16] studied the work fragmentation (through interruption) and related it to developers' productivity. They found that work fragmentation is correlated to lower observed productivity. Robbes and Röthlisberger [14] also analysed how developers' expertise is correlated to the effort on a task. They found a negative correlation between the time spent and the developers' experience. Soh *et al.* [18] analysed the complexity of the task resolution (*i.e.,* in the patch) and correlated it to the time spent on a task. They reported that the effort spent by a developer is not correlated with the implementation complexity. Bantelay *et al.* [1] combined ITs with commit data to calculate evolutionary couplings. They reported that the combined ITs and commit for change prediction achieved a maximum recall improvement of 13% and 2% maximum precision drop. Similarly, Zanjani *et al.* [23] improved an existing approach of impact analysis.

C. Interaction Traces for Recommendation

ITs have been used for code recommendation. TeamTrack [2] uses the association between previously-visited entities to recommend the next program entity to visit. NavTrack [17] uses only selection- and open-events on files to discover hidden dependencies between files and recommend files related to the file of interest. NavClus [8] improves the previous approaches by clustering navigational cycles. MI [9] includes view/selection histories of program entities in association rules and recommend entities that are not yet edited. Kersten and Murphy [5] used ITs to build task contexts and recommend program entities to developers. Their recommendation system is based on the principle of frequency and recency, *e.g.,* the more frequently and recently developers interacted with an entity, the more the entity is relevant. Robbes and Lanza [13] also used change history to propose a code completion tool that reduces developers' scrolling effort and that users found more accurate than the previous tools they used.

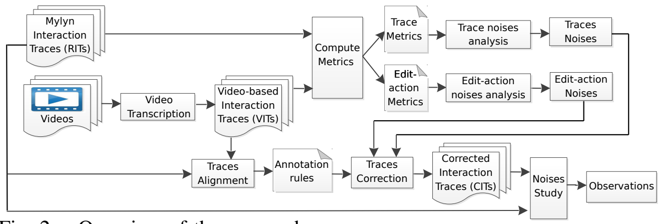


Fig. 2. Overview of the approach

Instead of studying developers' behaviour, development activities (and their relations), or providing a tool support based on ITs, our study investigates noises in ITs and assesses the effect of these noises on previous studies. Our study is related to the works above because it can provide new insights for a more reliable use of ITs and for the improvement of tools.

III. Noises in Interaction Traces

We now explain how we observe the noise in ITs and computed its characteristics. By noise, we refer to any events that may (1) have a 0 duration, that may (2) overlap with one another, and to edit-events that may not (3) correspond to real modifications of the source code. Noise can be at the trace-level and–or at the event-level: at the trace-level, the noise would impact the overall time spent on the task (see Section III-B) while at the event-level, the noise impacts the characterisation of edit events (see Section III-C).

Figure 2 summarises our approach. We first collect VITs and RITs. We collect RITs using Mylyn because we want to assess if these ITs contain any noise and Mylyn has been often used in previous studies. We collect VITs because they accurately capture the developers' interactions. Then, we compute and analyse metrics and report our observations on existing noises in RITs in comparison to VITs. Based on our observations, we propose an approach, in Section IV, to align RITs and VITs using rules to annotate events. Finally, we use our approach to correct RITs and study the impact of noise on some previous studies, in Section V.

A. Data Collection

We perform a controlled experiment to collect RITs and the video-captures necessary to obtain VITs [21].

1) Subject Systems: We choose four Java open-source systems[1] because we need their source code, which the partic-

[1]https://eclipse.org/ecf/, https://eclipse.org/pde/, http://www.jedit.org/, and http://www.jhotdraw.org/

ipants will modify to accomplish their tasks. We choose two Eclipse-based (plugin) systems (ECF and PDE) and two non-Eclipse systems (jEdit and JHotDraw). ECF (Eclipse Communication Framework) is a set of frameworks for building communications into applications and services. PDE (Plug-in Development Environment) provides tools to create, develop, test, debug, build, and deploy Eclipse plug-ins. JEdit is an open-source text editor. JHotDraw is a Java GUI framework for technical and structured Graphics. We choose these two kinds of systems because (1) we want to decrease threats to generalisability by using two Eclipse-based systems and two other non-Eclipse systems; (2) their source code is open; (3) they belong to different application domains; and, (4) they are well-known and studied in the software engineering community. We also choose these systems because there are RITs available in the bug reports of the two first ones while the other two have no relation with Mylyn.

2) Object Tasks: We seek concrete maintenance tasks. Thus, for each Eclipse-based system (ECF and PDE), we consider one of their bugs[2]: 202958 and 265931, respectively for ECF and PDE. We consider these bugs because (1) they are already fixed so we know that a solution exist and (2) these solutions can be implemented in reasonable times, about 45 minutes, which we estimated through a pilot study.

For the non-Eclipse systems, we choose one of their version randomly and define one task for each system so that each task requires around 45 minutes. The tasks were defined by exploring the two systems and identifying a possible need: in jEdit, to add lines number and error messages in the select line range dialog and, in JHotDraw, to change the colors of labels and background of the FontChooser.

3) Participants: We recruit participants via emails, which we send to the authors' research groups and individual contacts. We provide potential participants with a link[3] to an online form for registration, collecting information about their level of study, gender, numbers of years of Java and Eclipse experiences. We use this information to assign participants to systems so that participants with different profiles work on a same system to minimise threats to internal validity. To avoid learning bias, each participant perform only one task.

In total, 19 participants answered our emails. Out of 19, four did not complete the tasks because (1) three did not have enough Java knowledge and (2) one abandoned the experiment. Thus, we consider only 15 participants: 13 are Ph.D. students in the Department of Computer and Software Engineering of Polytechnique Montréal; one is M.Sc. student at the Czech Technical University in Prague; and, one is a professional software developer.

We perform a Kruskal-Wallis rank-sum test to assess whether the numbers of years of Java and Eclipse experience differ between groups of participants performing the tasks on different systems and found that the differences are not statistically significant (p-values of 0.67 and 0.96). Tasks are performed by participants with similar numbers of years of Java and Eclipse experience.

[2]https://bugs.eclipse.org/bugs/show_bug.cgi?id=202958 and id=265931

[3]http://goo.gl/DyQu6j

We use a checklist of the steps of the experiment to consistently provide the same (and only the same) information to each participant. We let each participant know before the experiment that she will perform one maintenance task on one Java system for about 45 minutes, although there is no time limit. We also inform her that the collected data is anonymous. We ask the participants to try their best to complete the tasks but also that they can leave the experiment at any time for any reason without penalty whatsoever.

4) Collected Data and Processing: Participants performed their tasks using the Eclipse IDE together with the Mylyn plugin to collect RITs. The subject systems were hosted in our SVN repository[4]. We imported the systems into the participants' environments and collected their RITs at the end of the tasks (as well as patches for future studies). To collect RITs, we create the task in the IDE. Mylyn starts collecting RITs when the participant activates the task. After the participant completes the task, she deactivates the task to stop collecting RITs. More details regarding how Mylyn collects RITs can be found in [5].

During the tasks, we also captured video recordings of the participants' screens using VLC Media Player 2.0.5[5] at 15 images per second. We transcribed the video captures into ITs manually. The first three authors defined a transcription template after viewing the videos once to minimise subjectivity. Then, they transcribed the videos into CSV files, including the following information. Finally, they constructed VITs from these CSV files. (All the data is available online[6].)

- Start (hh:mm:ss): The timestamp when the event starts.
- End (hh:mm:ss): The timestamp when the event ends.
- EntityName (string): The name of the entity concerned by the event.
- EntityType (project, package, or file): The type of the entity concerned by the event.
- Origin (Package Explorer, Outline View, Search Result, etc): The part of the IDE where the event was triggered.
- Technique (Visual, Static Relation, Text Search, Reference): The method used by the participants to move from the previous event to the current event. For example, "visual" is when the participant is visually searching a relevant entity by scrolling the package explorer or the code editor, then open a file through the package explorer. We consider that the participant reaches the opened file through visual search. "Static relation" is for example the navigation from a method call to its declaration.
- Activity (Selection, Open, Search, Run, Edit, Close, Other): The activity performed by the participants such as selecting a file (in the package explorer), opening a file, searching through a keyword, running the system, editing the code, or closing an opened file.
- Comments (string): Other information that the transcriber found useful for further analysis.

[4]http://goo.gl/Mskalh

[5]http://www.videolan.org/vlc/releases/2.0.5.html

[6]http://www.ptidej.net/downloads/replications/esem15/

TABLE I. p-VALUES OF THE COMPARISON BETWEEN GT AND AT, AND BETWEEN RITs AND VITs

	GT vs. AT		RITs vs. VITs		RITs vs. VITs	
	VITs	RITs	GT	AT	Idle	Overlap
ECF	0.88	0.02	0.68	0.02	0.02	0.02
jEdit	1	0.1	0.4	0.1	0.06	0.06
JHotDraw	0.68	0.02	1	0.02	0.02	0.02
PDE	0.77	0.05	0.68	0.05	0.02	0.02

B. Trace Level

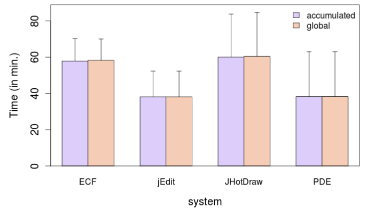
We compare both RITs and VITs to access all the entities involved in the ITs, the times spent and the activities performed on these entities, and the IDE views used by participants. At trace level, we study the possible noises introduced by 0-duration events and overlapping events because some researchers compute the times spent on tasks by participants considering the start and end timestamps of the whole ITs, *e.g.,* [14], while others aggregate and clean the times spent in each event, *e.g.,* [19].

1) Approach: We consider the times spent performing the tasks as defined below and illustrated in Figure 1, which is an IT involving three events e_1, e_2, and e_3. We study the mismatches between VITs time (T_{VITs}) and RITs time (T_{RITs}). We compute both T_{VITs} and T_{RITs} in two ways:

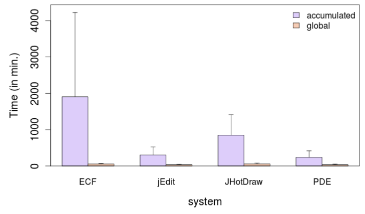
- Global time: we compute the global time using the start and end timestamps of the ITs as $GT = End(IT) - Start(IT)$. The global time of the IT in Figure 1 would be $GT = End(e_3) - Start(e_1)$
- Accumulated time: we compute the accumulated time in an IT as the sum of the times spent on each event: $AT = \sum_{event \in IT} End(event) - Start(event)$. The accumulated time of the IT in Figure 1 would be $AT = \sum_{e_i \in \{e_1, e_2, e_3\}} End(e_i) - Start(e_i) = d_1 + d_2 + d_3$

To assess if there are noises in the times computed from ITs, we first compare the global and accumulated time of VITs, on the one hand, and RITs, on the other hand. This comparison aims to figure out whether the two ways of computing times give the same results. If global and accumulated times are the same, to study the noises in time spent on task, we must only compare one of them between VITs and RITs. However, if they are different, we must compare both of them between VITs and RITs. The comparison of the times between VITs and RITs aims to assess the possible noises in RITs (compared to VITs). We use the two-sided Wilcoxon unpaired test to assess the differences stated above because we are not interested in the direction of the difference, only to know if there is a difference. We consider that the difference is significant at $\alpha = 0.05$,

2) Results and Discussions: Figure 3 presents the difference between GT and AT computed from VITs (See Figure 3(a)) and RITs (See Figure 3(b)). Regarding RITs, the difference is statistically significant for two systems (ECF and JHotDraw) and borderline for PDE system (p-value exactly 0.05), as shown in Table I (first part). This result suggests that the two ways of computing time from RITs do not provide the same results. Thus the way the time spent on a task is computed in the previous studies may affect the results found. Figure 4 presents the differences in the times spent performing the tasks between RITs (GT_{RITs}) and VITs (GT_{VITs}). Figure 4(a) shows that GT_{VITs} is higher than GT_{RITs}. On the contrary, AT_{RITs} is higher than AT_{VITs} in Figure 4(b).

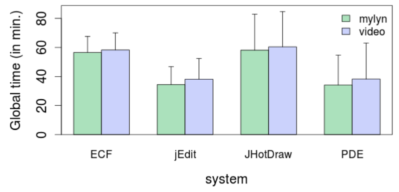


(a) Video (VITs) time

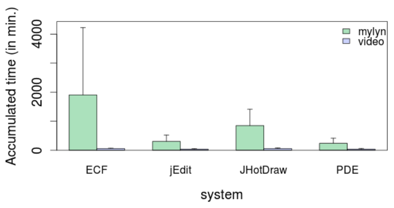


(b) Mylyn (RITs) time

Fig. 3. Difference in global and accumulated times



(a) Global time



(b) Accumulated time

Fig. 4. Difference in task resolution times between RITs and VITs

We studied whether the differences are statistically significant. Table I (second part) shows that for ECF and JHotDraw, there are statistical significant differences between AT_{VITs} and AT_{RITs}. For jEdit and PDE, the differences are not statistically significant, but are close, with p-values of exactly 0.05. The absence of significant differences for global times between RITs and VITs hints that global time is closer to the actual time spent. We expected this result because participants performed their tasks without interruptions. However, in real-work settings, the GT may not reflect the time spent on the tasks. We argue that in real-work settings, developers may interrupt their work and turn off the collection of the trace. Later, when they turn on the task (collecting traces), Mylyn reloads the previous trace and adds the collected activities. To support this statement, we observe in the RITs (from Bugzilla) some traces that ended a month or more after their start dates. So, the global time in this cases includes times during which developers did not work on the task. Thus, even without statistical significant differences between GT_{VITs} and GT_{RITs}, we claim that global times are not reliable for RITs gathered by developers in their daily work. Robbes and Röthlisberger [14] relate the development time to developers' expertise by considering the global time as the time spent performing a task (*i.e.,* "the difference between the timestamps of the last and the first events"). The global times seem reliable in our data-set because they come from a controlled experiment.

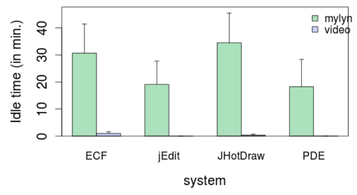
Moreover, the differences, even not statistically significant, observed in Figure 4(a) indicate a mismatch between GT_{VITs} and GT_{RITs}. We explain that this mismatch may be caused by the tool for gathering ITs and the delays between the developers' actions and the collection of the data. We observe that Mylyn starts collecting data when the participants interact with the first program entity, *e.g.,* if a participants starts the task by scrolling without interacting with any entity, Mylyn will not collect any data. This noise does not concern Mylyn only, any monitoring tool may be subject to this noise. Overall, RITs miss on average 2.91 minutes (about 6%). Hence the following observation.

Observation 1: *Mylyn ITs miss on average about 6% of the times spent during the session.*

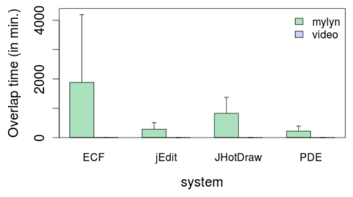
Figure 4(b) and Table I (column "AT" in second part) show that accumulated times are also different between RITs and VITs and that this difference is statistically significant. We assert that this observation comes from idle and overlap times.

We consider that there is an idle time between two consecutive events if there is a non-zero time period between these events, *i.e.,* the second event was not triggered immediately after the end of the first event. There is idle time (*it*) between the events e_2 and e_3 in Figure 1. The lack of interactions with the IDE (thinking, reading code or task description) and the interactions that the monitoring tool cannot handle (*e.g.,* scrolling) appear in RITs as idle. We cannot distinguish idle times from real inactivities that can happen in real-world settings. Thus, we name all the "inactivity" periods in the RITs as idle times. We consider that there is an overlap between two consecutive events if the second event starts before the end of the first event. There is overlap time (*ot*) between the events e_1 and e_2 in Figure 1. An overlap may occur because of the aggregation of the events and–or the activities involving several program entities (*e.g.,* selection of many files in the package explorer). Idles and overlaps affect the overall times spent to perform tasks. Yet, previous study did not consider idles and overlaps when computing global times, eg [14].

Figure 5 shows the cumulative durations of idles and overlaps. It shows that these phenomena are prevalent in RITs compared to VITs. It also shows that overlap times are more important in duration than idle times. Table I (third part) shows that the differences between idle and overlap times in RITs and VITs are statistically significant for three systems (ECF, JHotDraw and PDE) while it is closed to be significant (p-value = 0.06) for JEdit. Overall, RITs contain a average cumulative idle and overlap times of 26.08 (median 27.55) minutes and 579.3 (median = 250.4) minutes, respectively. While idle time can be seen as the time spent reading code, thinking or understanding the code, the overlap time seems too much. However, it can be due to the overlap between many events (multi-overlaps).

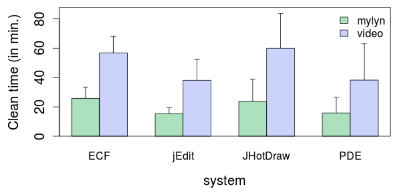


(a) Idle times

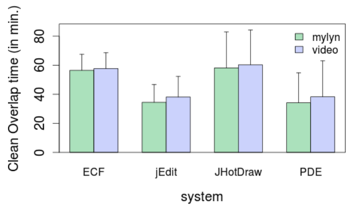


(b) Overlap times

Fig. 5. Difference in idle and overlap times between RITs and VITs



(a) Clean idle and overlap times



(b) Clean overlap times

Fig. 6. Difference in clean times between RITs and VITs

Observation 2: *Mylyn ITs involve on average about 53% of idle times and 1,171% of overlap times.*

When we recompute the accumulated times after removing idle and overlap times, as shown in Figure 6(a), the new accumulated times of RITs drop down, but is still different from the global time (See Figure 6(a) vs. Figure 4(a)). However, if we keep the idle times but remove only the overlap times (See Figures 6(b)), then the accumulated and global times are similar, both for VITs and RITs (See Figure 6(b) vs. Figure 4(a)). Thus, we consider that idle times that may appear in the RITs collected in a real-world setting are not always interruptions.

The last observation arises from our interest to quantify the magnitude of individual idle times. We investigate this magnitude because the cumulative durations of these idle times can reach an average of 26 minutes, and the data collected by developers in their daily work may involve real interruptions. Figure 7 (durations of individual idle) shows that idle times in

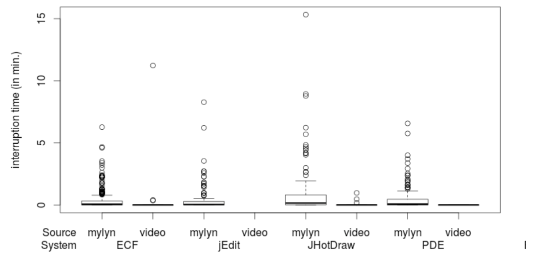


Fig. 7. Distribution of the time of individual idle times for RITs

RITs can reach more than five minutes. However, overall, idle times lasted 0.5 (median = 0.07) minutes.

Observation 3: *Mining times from ITs requires to remove overlap times from accumulated times. Idle times that last less or equal to half a minute should not be consider as interruptions and should be include in the time spent on task.*

By quantifying the average duration of idle events that are not interruptions, our result complements Sanchez *et al.* [16] who was inspired by previous study and defined interruptions in RITs as a pause of programming of duration greater or equal to 3 minutes. Our threshold of idle times is in the context of RITs *i.e.,* considering idle times within developers' activities. The threshold of interruption times defined by González and Mark [3] was found by considering more general workers' activities as the considered workers were four developers, six business analysts, and four managers.

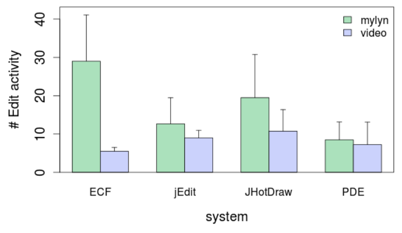
C. Event Level

At the event level, we focus on the noises for edit events because (1) edit events are more accurately identifiable from video data than other events as it is trivial to see in the videos if the code is being changed and (2) edit events are subject to several assumptions, such as being representative of the activity of changing source code [9], [16], [19]. Also, edit events have been used to build code recommendation tools [4], [9] and as proxy to productivity [5], [16]. Thus, any bias related to edit events would impact such previous studies.

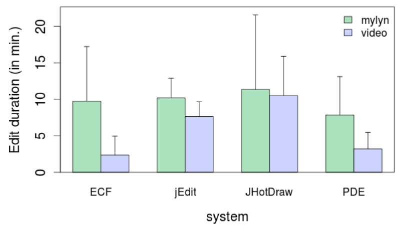
1) Approach: We consider the numbers of edit events and the times spent performing edit events to investigate noises in the identification of edit events, *i.e.,* any differences between VITs and RITs in numbers of edit events and the times spent performing edit events.

- Numbers of edit-events: we count the edit-events after removing events whose start and end timestamps are equal (*i.e.,* 0-duration), which were considered selection-events in previous studies [9], [16].
- Edit duration: we compute the edit durations after removing overlap times.

To evaluate the difference between VITs and RITs for the numbers of edit events and edit durations, we use the same statistical test as at trace level (Section III-B1).



(a) Numbers of edit-events



(b) Durations of edit-events

Fig. 8. Difference in edit-events between RITs and VITs

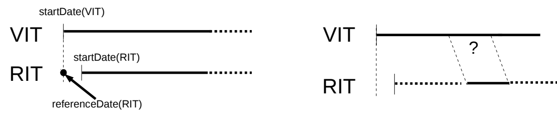
TABLE II. COMPARISON BETWEEN T_{VITs} AND T_{RITs} EDIT-EVENTS

	p-value	
	Number edit	**Edit duration**
ECF	0.13	0.26
jEdit	0.8	0.4
JHotDraw	0.34	1
PDE	0.66	0.34

2) Results: We observe differences in VITs and RITs according to their numbers of edit-events, see Figure 8(a), and the durations of the edit-events, see Figure 8(b). The differences are not statistically significant as revealed by the Wilcoxon test results from Table II. The numbers of edit events and the durations of the events mined from RITs are not statistically different to that from VITs.

The lack of statistical significant difference in the numbers and durations of edit events is the consequence of the task resolution *i.e.,* how successfully participants resolved the tasks. Indeed, participants performed few changes. Only 12 out of the 15 participants performed at least one edit event while only five participants successfully resolved their tasks (and three participants partially). Overall, eight participants (53.33%) changed the code during their task. As resolving the task requires to change the code, results of the statistical tests would be different if all participants perform changes.

However, even though they are not statistically different, the observed difference in Figure 8 reveals that there are some edit-events that are not really changes to the code (See Section IV-B for the details about these edit-events). We call these edit-events "false edit-events" and define the measure edit bias $edit\_bias = \frac{false\_edit}{real\_edit}$, where $false\_edit = edit(RITs) - edit(VITs)$ and $real\_edit = edit(VITs)$. Using the edit bias, we observe a bias of, on average about 28%, which contradicts the assumption that all non 0-duration edit events correspond to changes to the code.



(a) Reference start timestamps (b) Alignment principle

Fig. 9. Illustration of the alignment of RITs and VITs

> ***Observation 4:*** *Interaction traces contain about 28% of false edit-events.*

IV. Approach to Annotate Events

We now propose an approach to correct RITs by identifying false edit-events.

A. Approach

We align VITs with RITs to generate corrected ITs (CITs) corresponding to RITs. The alignment consists in identifying VITs events corresponding to RITs events. We use the timestamps as keys for traces alignment. First, we manually identify the *RITs start timestamps reference* ("reference timestamps" in the remainder), which is the RITs timestamps corresponding to the start timestamps of VITs, see Figure 9(a). Timestamps are in different format and have different precision. RITs timestamps are more precise (in term of milliseconds) than reference timestamps (in term of minutes). This difference in precision could affect the alignment of the events but, given the durations of the events, a precision in minutes is sufficient.

Second, we use the reference timestamps to compute the exact time when an event starts and ends. For example, consider that an event e started at timestamp d_1 and ended at timestamp d_2. We compute the start and end time of the event e as $startTime(e) = time(d_1) - time(d)$ and $endTime(e) = time(d_2) - time(d)$ where d is the reference timestamp (*i.e.,* the timestamp corresponding to the start of the video recording) and $time(x)$ is the number of milliseconds[7] to reach the timestamp x. Having the start and end time of each event in a VIT and a RIT, we can align two events if they (ideally) start at the same time and end at the same time. However, the alignment is not always ideal. To avoid wrong alignments due to overlap times between events, we remove overlap time intervals before aligning the events, *e.g.,* for consecutive events e_1 and e_2, if $End(e_1) > Start(e_2)$, we consider that $End(e_1) = Start(e_2)$. This cleaning does not affect the start time of an event which the alignment is mostly based on. The alignment consists to search the events in a VIT corresponding to each event in RITs, see Figure 9(b). For each event in RITs, we scan the VITs and search for the corresponding event(s).

We consider that an event in a RIT is aligned to an event in a VIT if the intersection of their time period is not empty. For example, if an edit event e in a RIT is aligned with n events $\{e_1, e_2, ...e_n\}$ in a VIT, we consider that the edit event e is aligned to the first edit event in $\{e_1, e_2, ...e_n\}$. We could use DTW (Dynamic Time Warping) for alignment, but our goal is not to minimise the distance between RITs and VITs as does DTW. Instead, we align vertically events and identify matching events in a RIT and in its corresponding VIT.

[7]We use the method getTime() of the Java Date class.

TABLE III. Rules for annotation of edit events

RITs events	Times	Previous event	Annotated events
Edit	$time \leq 24.01sec$	Search view	Open
Edit	$time \leq 24.01sec$	Other	Navigation
Edit	$time > 24.01sec$	N/A	Edit

We manually check the alignments of some randomly-selected ITs and identify the VITs events that correspond to RITs events and build the following rules (see Section IV-B) to annotate RITs events. We randomly choose some ITs because checking all the ITs could not impact negatively our approach. On the contrary, checking all ITs could only lead to more fine-grained annotations that would only be sub-categories of the current annotations.

B. Results

Through the alignment, we identified VITs events that correspond to RITs events. RITs involved two main kinds of events: selection and edit. For edit-events, we observed that false edit-events with not 0-duration take on average 24.01 seconds (median = 4.14), while true edit-events take on average 143.7 seconds (median = 114.4). Using both the mean and third quartile (28.33 seconds) of the times of false edit-events as thresholds does not yield major differences on the recall of false edit-events. Thus, we assume that:

> ***Observation 5:*** *An edit-event is a false edit-event if it takes less than 24.01 seconds but more than 0 second.*

When looking at the alignments, most of the false edit-events correspond to the opening of a file (*e.g.,* double click to open the file), the navigation of static relation between program entities and the text-based search in the file. When we look at RITs, we did not find any indicator to distinguish which false edit-event corresponds to each of the VITs event above. However, one case of open event was identifiable, *e.g.,* a false edit-event after an event triggered through the search view. We build our correction approach with the annotation rules in Table III. For example, the first row in the table means that if an event is an edit event that lasted less than or 24.01 seconds and if the previous event was triggered through a search view, the event should be considered an open event.

Regarding selection events, the alignment reveals that selection events can occur through different views (package explorer, outline view, search view, editor). While RITs equally considers these selection events, the video captures show that selection events in the editor or type hierarchy views (compare to selection events through other views) are performed when the participants expect the entities to be relevant. This observation is consistent with Ko *et al.* [6], who stated that the selection in the editor view indicates a perception of relevance.

Hence, a selection event in the editor and type hierarchy views is likely to indicate the relevance of an entity, we consider the selection of an entity in the editor as the inspection of the entity because the participants seem to investigate the entity to see if it is relevant or to understand it. Thus, we annotate the selection event as indicated in Table IV.

Our annotation rules (Tables III and IV) provides a way to correct RITs. Some edit-event really should be navigation- and open-events while some selection-events really should

TABLE IV. RULES FOR ANNOTATION OF SELECTION EVENTS

RITs events	Origin	Annotated events
Selection	Editor	Inspection
Selection	Other	Selection

TABLE V. EDITING STYLES BETWWEN VITS, RITS, AND CITS

	VITs	RITs			CITs		
		#	P(%)	R(%)	#	P(%)	R(%)
edit-first	1	0	0	0	0	0	0
edit-last	9	1	100	11.11	3	100	33.33
edit-throughout	2	11	18.18	100	9	22.22	100

be inspection-events. We present a preliminary study of the possible effects of the correction in Section V.

V. WHAT IS THE EFFECT OF THE NOISES AND ITS CORRECTION ON PREVIOUS WORK?

We now assess whether applying our approach impacts the results from two previous studies. We consider the study on editing style by Ying and Robillard [22] and the study on exploration strategies by Soh *et al.* [19]. We choose these two studies because (1) their use of ITs can be affected by the noises observed in Section III, and (2) we have access to their data and–or scripts.

We revisit these previous studies by applying their approaches on two data-sets. The first data-set contains the RITs and VITs collected during our experiment in Section III. The second data-set contains ITs collected from bugs reports in Bugzilla for the ECF, Mylyn, PDE and Platform, called bRITs in the following and containing 1,970 ITs. We correct RITs and bRITs using the approach described in Section IV to obtain CITs and bCITs, respectively.

A. *Effect of Noises on Editing Styles*

We use the script[8] provided by the authors of the study to compute the editing styles [22]. We use precision and recall to compare the editing styles identified in, on the one hand, RITs, VITs, and CITs, considering VITs as oracles and, on the other hand, bRITs and bCITs.

$$Precision(s) = \frac{\text{\# traces correctly identified as style } s}{\text{\# traces identified as style } s}$$
$$Recall(s) = \frac{\text{\# traces correctly identified as style } s}{\text{\# traces of style } s}$$

where $s \in \{\text{edit-first}, \text{edit-throughout}, \text{edit-last}\}$

We also compute the numbers of miscategorisation between ITs, *e.g.,* when an IT categorised as *edit-first* (using bRIT) becomes *edit-throughout* or *edit-last* (using the corresponding bCITs), which we note "style X → others" with $styleX \in \{\text{edit-first}, \text{edit-throughout}, \text{edit-last}\}$. Similarly, an IT may change the editing style from other style (*e.g., edit-throughout* or *edit-last*) to a style X (*e.g., edit-first*).

1) Results: Table V presents the results of identified editing styles for VITs, RITs and CITs for the 12 participants who performed at least one edit event in our experiment. The table shows that one, nine, and two traces are identified in VITs as *edit-first*, *edit-last*, and *edit-throughout*, respectively. Compared to editing styles identified in RITs, nine traces $(11-2)$ are wrongly identified as *edit-throughout*. The nine traces are one *edit-first* and eight *edit-last*. Similarly, seven traces $(9-2)$ are wrongly identified as *edit-throughout* using CITs. The seven traces are one *edit-first* and six *edit-last*. These results shows that both RITs and CITs contains noises that affect the identification of editing styles. However, when comparing RITs with CITs, using CITs improves the precision and recall by about 22% (*edit-last*) and 4% (*edit-throughout*).

[8]http://www.cs.mcgill.ca/aying1/2011-icpc-editing-style-data.zip

TABLE VI. EDITING STYLES BETWEEN BRITS AND BCITS

	bRITs	bCITs	bRITs ∩ bCITs	style → others	others → style
edit-first	163	521	142	21 (1.06%)	379 (19.23%)
edit-last	546	438	279	267 (13.55%)	159 (8.07%)
edit-throughout	1,261	1,011	872	389 (19.74%)	139 (7.05%)
All	1,970	1,970	1,293 (65.63%)	677 (34.36%)	677 (34.36%)

Table VI shows similar trends for bRITs and bCITs. About 66% of the ITs conserve their editing style in both bRITs and bCITs (column "bRITs ∩ bCITs") while editing styles change for 34% (column "style → others"). These results show that our approach (modification of traces) yields in some cases different categorisation of ITs and, hence, that noise impacts the results of this previous study.

2) Discussions: Using our experiment data-set and approach shows a small difference in the identification of editing styles. We use the categorisation threshold from the original work, which is based on the percentage of edit-events in the first half of an IT [22]. The considered threshold was inferred from a large set of ITs. We explain the small difference on the experiment data-set by the small size of the data-set as the threshold used may not be suitable for this amount of IT. However, the use of Bugzilla data-set, which may be more suitable to the threshold due to the size of the data-set, shows about 34% of changes in editing styles.

As example of differences between ITs, we observe that the trace of participant S06 is categorised as *edit-first* when using VITs because, as expected, it has only edit-events in its first half. The corresponding trace has 29% and 50% of edit-events when considering RITs and CITs, respectively. We assumed that applying the annotation rules, which improves the categorisation of RITs wrt. VITs for the experiment data-set would also improve the categorisation of RITs from the Bugzilla data-set. We cannot verify this assumption because we do not know the true categorisation for the Bugzilla dataset.

B. *Effect of the Noises on Edit Ratio and Time Spent*

To assess the effect of noise on edit ratio and time spent used by Soh *et al.* [19] to evaluate exploration strategies, we compute the number of edit-events in an IT divided by the total number of events in RITs, VITs, CITs as well as bRITs and bCITs and compare these ratios using the Wilcoxon unpaired test. We consider that a difference is significative at $\alpha = 0.05$. We also compute the durations of the ITs to obtain the times spent by participants on the tasks. We include idle times (less or equal to 0.5 min) in the time spent on the tasks as discussed in Section III-B2. Thus, for VITs, we consider their global times while, for RITs, CITs, bRITs, and bCITs, we consider the accumulated times of all the events after removing overlap and adding idle times. We use the Spearman coefficient to

TABLE VII. EDIT RATIOS BETWEEN VITS, RITS, AND CITS

	p-values		
	VITs vs. RITs	VITs vs. CITs	RITs vs. CITs
ECF	0.02	0.46	0.02
jEdit	0.1	1	0.1
JHotDraw	0.02	0.68	0.02
PDE	0.02	0.68	0.11

TABLE VIII. SPEARMAN COEFFICIENT BETWEEN EDIT RATIOS AND TIMES SPENT

(a) VITs, RITs, and CITs

	Spearman correlations		
	VITs	RITs	CITs
ECF	0.31	-0.4	0.6
jEdit	-0.5	0.5	-1
JHotDraw	-0.4	0.8	1
PDE	0.2	0.8	1

(b) bRITs and bCITs

	Spearman correlations	
	bRITs	bCITs
ECF	0.11	0.6
Mylyn	0.07	0.52
PDE	0.30	0.66
Platform	0.08	0.60

assess the relation between edit ratios and the times spent on tasks. This relation is evaluated because Soh *et al.* [19] observed that edit ratio and the time spent was related to different exploration strategies, *i.e.,* edit ratios and times spent were not correlated.

1) Results and Discussions for Edit Ratios: The analysis of the edit ratios shows significant difference between VITs and RITs, except for jEdit, in Table VII (first column). The differences between VITs and CITs are not statistically significant for all systems, see Table VII (second column). While the differences between RITs and CITs are system-dependent for the experiment data-set, in Table VII (third column), the Bugzilla data-set shows significant differences for all systems.

The significant differences between VITs and RITs show that using RITs may result in inaccurate findings. Applying our correction approach reduces the mismatch between VITs and RITs as there are differences between VITs and RITs but not between VITs and CITs. The system-dependent differences between RITs and CITs could indicate that our approach may not fully address the noises in RITs or that other characteristics of the VITs and RITs reduce the efficiency of our approach. Future works include studying in details this observation.

2) Results and Discussions for Times Spent: Tables VIII show the relations between edit ratios and times spent. Computing the correlation on VITs shows that the edit ratios tend to be weak or inversely correlated with the times spent, Table VIII (a), first column while on RITs and bRITs, it shows positive correlations (except for the ECF system), second column of Table VIII (a) and first column of Table VIII (b). After applying our correction approach, the correlations increase, except for jEdit system in the experiment data-set.

The correlations between edit ratios and times spent in VITs have no common trend, while those in RITs are positive for 3 out of 4 systems. For the systems that have a common trend, we observe that the correlation is more pronounced for RITs than for VITs. The corrected ITs, which show positive correlations (except for jEdit), indicate that the more time is spent by participants, the more they perform edit events over other events. Contrary to the study of Soh *et al.* [19] that may indicate the absence of correlation between edit ratio and the time spent on task, the result of CITs is realistic because the changes of code may take more time than other activities. However, the observed contradiction with the result of VITs (no correlation) can be explained by the number of events in RITs. In fact, while the transcription of the videos (*i.e.,* VITs) resulted in more events than RITs, CITs correct RITs and have the same number of edit-events as VITs, but the total number of events is less than those of VITs. This difference in the number of events increase the edit ratios of CITs that may correlated with the time spent.

VI. THREATS TO VALIDITY

Our results may suffer from threats to validity.

Interaction traces: We use Mylyn plugin to collect ITs. As many other monitoring tools (*e.g.,* Mimec) can collect ITs, we cannot guarantee that Mylyn may provide the most accurate ITs. However, Mylyn is the most used tool for collecting ITs in industrial projects and our experience with Mimec shows that its ITs have the same noises. Another threat when using Mylyn is the version of the tool. During our experiment, we use different versions of Eclipse and, consequently, different versions of Mylyn for different tasks. We were constrained to use specific versions of the Eclipse-based systems (*i.e.,* ECF, PDE) because we aimed to use the versions for which the bugs were reported. However, as features are not significantly different between versions of Mylyn, the differences between the ITs of two versions of Mylyn are negligible. The overlap phenomenon can also affect our results. We consider the overlap between two consecutive events but not among several events, which could affect our results.

Transcription of the video: The first three authors independently translated the videos into ITs. As some actions may be difficult to interpret or the exact start and end timestamps of an action may be difficult to identify, different transcriptions of a video by two authors may provide different results. However, to mitigate this threat, we defined a common template for video transcription. Moreover, the noises studied in this paper are related to the time and the edit events and edit events are easy to identify on a video capture.

Alignment of Mylyn and video ITs: To identify the mismatch between Mylyn and video capture, we align both their ITs. The alignment consists of identifying the events corresponding to one another in the two traces. As the alignment is based on the times when the events occur, any shift may result in the lost of alignment between two events that are actually matching. To mitigate this threat, we manually checked some alignments and observed that the false edit-events, for example, were not due to shifts in alignments.

Correction of ITs: We use the threshold inferred from our experiment to annotate edit-events. The time spent on false edit-events may be different for other systems and–or other developers (*e.g.,* developers' experience and speed when triggering events). To mitigate this threat, we apply the correction approach on RITs from our experiment and Bugzilla and observed the same trend.

Task and systems: The participants performed maintenance tasks on Java systems only. Hence, we cannot guarantee that using other systems and other programming languages may yield the same results. We choose the tasks to be accomplished in about 45 min. More complex tasks would take more time and developers would have more interruptions and would switch more often between tasks. Therefore, the tasks considered in our experiments may not be representative of tasks performed by developers in industrial settings. However, as we aimed to assess possible noises, we must have an oracle against which to compare collected ITs. Thus, we were constrained to collect ITs in an experimental setting.

Reliability: The reliability threat relates to the possibility of replicating our study. We mitigate this threat by providing all the data online. For the manual work required to replicate our study, we define a video transcription template and three of the authors transcribed the videos. Thus, we think that the transcription by other peoples may not significantly affect the results (*e.g.,* identifying edit-events from the video is not challenging). Moreover, we automated the alignment of events between RITs and VITs. Our tool can be used after video transcription without any manual work if the replication setting allows videos and RITs collection to start at the same time (*i.e.,* no need to manually identify the reference timestamps).

VII. Conclusion

Developers' interaction traces (ITs) have been widely used for several purposes, *e.g.,* assessing how developers perform maintenance tasks, and recommend program entities to edit. However, ITs have been subjects to several assumptions, *e.g.,* edit events are due to be code change actions, and the times mined from ITs are assumed to be the times spent by developers to perform the maintenance tasks. Yet, ITs collected in non-controlled settings may be subject to noises. Video captures of the screen of developers have been used as an alternative way to study how developers maintain and evolve software systems.

In this paper, we examine possible noises in ITs and the possible effects of the noises on two previous works. To achieve this goal, we conducted an experiment with 15 participants, whom we asked to perform some maintenance tasks. We collected both ITs and videos captures of participants' screens. We compared the ITs and videos and observed that ITs may miss up to 6% of the times spent performing tasks and that they may contain about 28% of false edit-events. By looking in details at the mismatches between these two data sets (*i.e.,* ITs and videos), we proposed an approach based on rules to annotate ITs to reduce the observed noises. We corrected ITs from both our experiment and Eclipse Bugzilla issue tracker system. Applying our approach on two previous works [22], [19], we report that these noises may have led researchers to mislabel some participants' editing styles in 34% of the cases and to consider that the numbers of edit-events performed by developers and the times that they spent on tasks are correlated, when they were considered not to be. Our results suggest that the observed noises must be considered carefully when mining interaction traces and building recommendation tools based on collected ITs.

Our future work consists of using the proposed correction approach and its annotation rules to build association rules that may help to improve the recommendations of program entities using ITs. We will also extend our study with more complex tasks and investigate other possible noises in ITs (*e.g.,* the noises in the program entities involved in the ITs).

References

[1] F. Bantelay, M. Zanjani, and H. Kagdi. Comparing and combining evolutionary couplings from interactions and commits. In *Proceedings WCRE*, pages 311–320, Oct 2013.

[2] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 241–248, 2005.

[3] V. M. González and G. Mark. "constant, constant, multi-tasking craziness": Managing multiple working spheres. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 113–120, 2004.

[4] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *Proceedings of the 4th International Conference on Aspect-oriented Software Development*, AOSD '05, pages 159–168, 2005.

[5] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT/FSE*, pages 1–11, 2006.

[6] A. Ko, B. Myers, M. Coblenz, and H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971–987, Dec 2006.

[7] L. M. Layman, L. A. Williams, and R. St. Amant. Mimec: Intelligent user notification of faults in the eclipse ide. In *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '08, pages 73–76, 2008.

[8] S. Lee and S. Kang. Clustering navigation sequences to create contexts for guiding code navigation. *Journal of Systems and Software*, 2013.

[9] S. Lee, S. Kang, S. Kim, and M. Staats. The impact of view histories on edit recommendations. *Software Engineering, IEEE Transactions on*, 41(3):314–330, March 2015.

[10] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the eclipse IDE? *IEEE Software*, 23(4):76–83, July 2006.

[11] Mylyn. http://eclipse.org/mylyn/.

[12] Mylyn Integrator Reference.
http://wiki.eclipse.org/index.php/mylyn/integrator_reference.

[13] R. Robbes and M. Lanza. Improving code completion with program history. *Automated Software Engineering*, 17(2):181–212, June 2010.

[14] R. Robbes and D. Röthlisberger. Using developer interaction data to compare expertise metrics. In *Proceedings MSR*, pages 297–300, 2013.

[15] M. Robillard, W. Coelho, and G. Murphy. How effective developers investigate source code: an exploratory study. *Software Engineering, IEEE Transactions on*, 30(12):889–903, Dec 2004.

[16] H. Sanchez, R. Robbes, and V. M. Gonzalez. An empirical study of work fragmentation in software evolution tasks. In *Proceedings SANER*, pages 251–260, 2015.

[17] J. Singer, R. Elves, and M. A. Storey. Navtracks: Supporting navigation in software maintenance. In *Proceedings ICSM*, pages 325–334, 2005.

[18] Z. Soh, F. Khomh, Y.-G. Gueheneuc, and G. Antoniol. Towards understanding how developers spend their effort during maintenance activities. In *Proceedings WCRE*, pages 152–161, Oct 2013.

[19] Z. Soh, F. Khomh, Y.-G. Gueheneuc, G. Antoniol, and B. Adams. On the effect of program exploration on maintenance tasks. In *Proceedings WCRE*, pages 391–400, Oct 2013.

[20] J. Wang, X. Peng, Z. Xing, and W. Zhao. How developers perform feature location tasks: a human-centric and process-oriented exploratory study. *Journal of Software: Evolution and Process*, pages 1193–1224, 2013.

[21] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.

[22] A. Ying and M. Robillard. The influence of the task on programmer behaviour. In *Proceedings ICPC*, pages 31–40, june 2011.

[23] M. B. Zanjani, G. Swartzendruber, and H. Kagdi. Impact analysis of change requests on source code based on interaction and commit histories. In *Proceedings MSR*, pages 162–171, 2014.

[24] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan. An empirical study of the effect of file editing patterns on software quality. In *Proceedings WCRE*, pages 456–465, 2012.