Noname manuscript No.
(will be inserted by the editor)

Noise in Mylyn Interaction Traces and its Impact on Developers and Recommendation Systems

Zéphyrin Soh · Foutse Khomh · Yann-Gaël Guéhéneuc · Giuliano Antoniol

Received: date / Accepted: date

Abstract

Context: Interaction traces (ITs) are developers' logs collected while developers maintain or evolve software systems. Researchers use ITs to study developers' editing styles and recommend relevant program entities when developers perform changes on source code. However, when using ITs, they make assumptions that may not necessarily be true.
Objective: This article assesses the extent to which researchers' assumptions are true and examines noise in ITs. It also investigates the impact of noise on previous studies.
Method: This article describes a quasi-experiment collecting both Mylyn ITs and video-screen captures while 15 participants performed four realistic software maintenance tasks. It assesses the noise in ITs by comparing Mylyn ITs and the ITs obtained from the video captures. It proposes an approach to correct noise and uses this approach to revisit previous studies.
Results: The collected data show that Mylyn ITs can miss, on average, about 6% of the time spent by participants performing tasks and can contain, on average, about 85% of false edit events, which are not real changes to the source code. The approach to correct noise reveals about 45% of misclassification of ITs. It can improve the precision and recall of recommendation systems from the literature by up to 56% and 62%, respectively.
Conclusion: Mylyn ITs include noise that biases subsequent studies and,

Zéphyrin Soh
Ptidej Team, Polytechnique Montréal, Canada, E-mail: zephyrin.soh@polymtl.ca

Foutse Khomh
SWAT Lab, Polytechnique Montréal, Canada, E-mail: foutse.khomh@polymtl.ca

Yann-Gaël Guéhéneuc
Ptidej Team, Polytechnique Montréal, Canada, E-mail: yann-gael.gueheneuc@polymtl.ca

Giuliano Antoniol
Soccer Lab, Polytechnique Montréal, Canada, E-mail: giuliano.antoniol@polymtl.ca

Table 1: Example of Mylyn Events

StartDate	EndDate	OriginId	StructureHandle	Kind
2013-12-19 16:43:37.818 EST	2013-12-19 16:43:40.101 EST	PackageExplorer	SelectionManager.java	selection
2013-12-19 16:51:00.189 EST	2013-12-19 16:55:43.956 EST	CompilationUnitEditor	SelectLineRange.createFieldPanel()	edit

thus, can prevent researchers from assisting developers effectively. They must be cleaned before use in studies and recommendation systems. The results on Mylyn ITs open new perspectives for the investigation of noise in ITs generated by other monitoring tools such as DFlow, FeedBag, and Mimec, and for future studies based on ITs.

Keywords Software maintenance · Mylyn Interaction traces · Noise · Editing behaviour · Recommendation systems

1 Introduction

Developers must find where and how to change relevant program entities to successfully perform software evolution. They usually interact with program entities through their Integrated Development Environments (IDEs). They perform several kind of activities in their IDEs, such as opening files, navigating static relations, searching program entities, changing the code. These activities form a valuable source of information to understand developers' behaviour, to develop recommendation systems, and to improve productivity [10].

Developers' activities are recorded in the form of developers' logs or developers' interaction traces (ITs). ITs have been used to study developers' preferred views in an IDE [19], editing styles [36, 38], and exploration strategies [31] as well as work fragmentation [27], maintenance effort [23, 30], and to predict changes [2] and their impact [37]. ITs have also been used to provide code completion [22] and recommendations of program entities [9, 16, 17].

ITs are ordered lists of events triggered by developers while performing their tasks. Each event has a kind (*e.g.,* command, edit, selection), has a start and end timestamps, and is triggered on a program entity (*e.g.,* project, package, file, class, field, or method). Table 1 shows an example of two events collected by Mylyn. The first event shows the start and end timestamps and indicates that the developer triggered a selection event on the file `SelectionManager.java` through the `PackageExplorer`. The second event shows an edit event on the method `SelectLineRange.createFieldPanel()` through the compilation-unit editor (*i.e.,* `CompilationUnitEditor`).

ITs are usually collected by monitoring tools. Many monitoring tools exist to collect ITs, such as Blaze [7], DFlow [18], FeebBaG [1], Mimec [15], Mylyn [20], and WatchDog [3]. However, they differ in their goals and the exact data

that they collect. Developers use these monitoring tools to collect and provide[1] ITs to researchers.

Hence, ITs depend on the developers' tasks but also on developers' work processes. Indeed, ITs will record any and all events preformed in the IDEs, even if these events are not directly related to the tasks. The timestamps of the events will reflect any and all interruptions that the developers receive. Yet, researchers use ITs assuming that (1) the times mined from ITs are the times spent by developers to perform their tasks [23, 31] and (2) edit events are modifications of the code [17, 36, 38].

This article shows that these assumptions are not true and describes an approach to clean ITs, which could help the software-engineering community to better assist developers. It uses Mylyn ITs because (1) Mylyn is the tool most used for collecting ITs in industrial projects and (2) Mylyn ITs are available and have been used by other researchers in previous work. For the sake of simplicity, we use "interaction traces" and "ITs" in the following to refer to Mylyn interaction traces.

In our previous work published at the 9th International Symposium on Empirical Software Engineering and Measurement (ESEM) [29], we studied noise in ITs and we proposed a *threshold-based approach* to correct false edit events in ITs, *i.e.,* edit events that are considered to be changes while they are not. In this article, we propose a *prediction-based approach* to improve the correction of false edit events. We also use the data-set from Lee *et al.* [17] to investigate how noise affects the recommendation of program entities. Thus, this article answers the following research questions:

RQ1: *Is there any noise in Mylyn interaction traces?*
We conduct a quasi-experiment and collect both Mylyn RITs (Raw ITs) and video captures. We transcribe the video captures into VITs (Video-based ITs). We compare RITs and VITs and observe that RITs miss on average about 6% of the time spent by participants on their tasks and contain about 85% of false edit events.

RQ2: *How can we address the noise in Mylyn interaction traces?*
First, we align RITs and VITs to identify VITs events corresponding to RITs events. We observe that false edit events take on average about 24 seconds. Second, we propose a threshold-based approach consisting of rules to automatically generate CITs (Corrected ITs) corresponding to RITs. Third, we model the prediction of false edit events and derive a prediction-based approach for correcting RITs, which outperforms the threshold-based approach by about 5% precision (98% vs. 93%) and 34% recall (98% vs. 64%) when classifying false edit events.

RQ3: *What is the effect of the noise on editing styles and exploration strategies?*
Developers' editing styles model *when* developers edit their code during their tasks [36] while their exploration strategies model *how* they explore

[1] Example of IT shared in December 2015 by a developer when fixing a bug: `https://bugs.eclipse.org/bugs/show_bug.cgi?id=483421`

their code [31]. By applying our threshold-based noise correction approach, we observe that the edit ratio is correlated with the time spent by developers on their tasks, contrary to what has been reported in previous studies of exploration strategies. We also report that noise in ITs led researchers to mislabel editing styles for about 45% of the developers ITs.

RQ4: *What is the effect of noise on recommendation systems?*
Recommendation systems can use edit events to recommend program entities to developers during their tasks [17]. By applying our threshold- and prediction-based noise correction approaches, we revisit the work by Lee *et al.* [17] and report that correcting noise in ITs can improve the precision and recall of the recommendations by up to 56% and 62%, respectively.

In our previous work [29], we identified and quantified time-related and edit-related noise in ITs collected by Mylyn; proposed a threshold-based approach to correct edit-related noise; and assessed how noise affect edit-ratios [9, 27, 31] and the classification of developers' editing styles [36]. With this article, we propose a prediction-based approach to improve the correction of false edit events; compare the threshold- and prediction-based approaches on editing styles and edit-ratios; evaluate the effect of edit-related noise on the accuracy of a recommendation system.

The rest of the article is organized as follow: Section 2 provides background details. Section 3 to Section 6 answer our four research questions followed by a discussion of threats to their validity in Section 7. Section 8 summarises the related work. Section 9 concludes and proposes future works.

2 Background

We now present background information about interaction traces and noise and the recommendation approaches based on interaction traces.

2.1 Interaction Traces and Noise

An interaction trace is a set of interaction events. An interaction event records an action performed by a developer in the IDE. The information contained in an interaction event includes its start (*i.e., startDate*) and end (*i.e., endDate*) timestamps, the program entity on which it occurred (*i.e., StructureHandle*), the part of the IDE interface used to generate it (*i.e., originID*), its kind (*i.e., kind*), and its Degree Of Interest (DOI) [9], which measures the degree to which the program entity is relevant to the task at hand.

Researchers use start and end timestamps to compute the time spent on a task. They use the kind of event to compare edit vs. other actions, *e.g.,* selection. Consequently, noise in interaction traces divides into *time-related noise* and *edit-related noise*. Time-related noise may come from delays between a developer's first activity and the first event recorded in the IT, idle times/interruptions between activities not reflected in the events, aggregation

of events by the monitoring tool. Edit-related noise are due to edit events that may not correspond to real modifications to the source code. We define *Mylyn edit events* as events in Mylyn ITs that are labeled as edit events. These events may be *true edit events* when they correspond to real modifications of the source code or *false edit events* when they do not.

Table 2 summarises the two kinds of noise, their possible sources, and their consequences. The main sources of noise in ITs are:

- The intent of the collected ITs: in some studies, authors use ITs for purposes for which they were not intended. The intent of ITs collected by Mylyn is to reduce the information overload in the developers' IDEs. Mylyn uses the collected ITs to identify program entities that are relevant to the task at hand. Then, it filters the project explorer view of the IDE and shows only relevant program entities. Mylyn considers a program entity relevant if developers spent a certain amount of time on the entity in the editor, by selecting some text, for example. It labels these non-edit activities as edit events to increase the relevance of a program entity. It thus may introduce edit-related noise as these edit events are not true modifications of the source code. Mylyn also aggregates events when computing their degree of interest when they are triggered on a same program entity several times in a row. The timestamps of the resulting, aggregate event encompasses the timestamps of the aggregated events even if developers performed other activities between these events.
- The inherent limit of logging: ITs are collected when developers trigger events on program entities. For example, when developers start their tasks by navigating in the package explorer or by searching without interacting with any program entity, Mylyn cannot collect related events, which causes delays between the start times of the tasks and the times of the first recorded events. Moreover, as any tool that monitor developers' activities, Mylyn cannot collect developers' cognitive activities, such as thinking and reading, which yield to idle times in Mylyn ITs. The delays and idle times introduce time-related noise when computing the times spent on tasks. Mylyn also defines several levels of relevance that map to each kind of events, with edit events being at the highest level. Mylyn considers both edit event and the selection of text in the editor as highly relevant to the degree of interest of a program entity.

2.2 Interaction Traces and Editing Styles and Exploration Strategies

An editing style captures how a developer edits code during a programming session [36]. An editing style is induced from and qualify an interaction trace. There are three types of editing styles based on the fraction of edit events at different points in the ITs: *edit-first*, *edit-last*, and *edit-throughout*. An IT is categorized as edit-first if a high fraction of edit events occurs in the first half of the trace. The edit-last as if a lower fraction of edit events occurs in the first half of the IT, and edit-throughout if the IT is not edit-first or edit-last.

Table 2: Kind of Noise, their Sources and their Consequences

Noise	Causes		Consequences
	Intent of collected ITs	**Inherent limit of logging**	
Edit-related noise	Misinterpretation of edit events (actual use is different from the intended use)	Mislabeling of different high level of relevance	false positive of edit events
Time-related noise	Misuse of timestamp (overlap of events sometimes due to aggregation of events)	Inability to log some activities (delay — no interaction with program entities, thinking, reading)	under/over estimation of the time

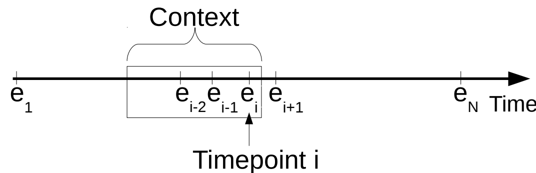


Fig. 1: Illustration of Recommendation based on Interaction Traces

An exploration strategy describes how developers navigate through program entities [31]. In a previous work, we studied exploration strategies and identified two types of strategies: referenced and unreferenced explorations. With referenced explorations, developers radiate from one set of files and keep on returning to the same set while, with unreferenced explorations, developers explore various seemingly unrelated files. We found that unreferenced explorations require more effort but are less time consuming than referenced explorations.

2.3 Interaction Traces and Recommendation Systems

A typical recommendation approach [17] includes the following steps: (1) mining interaction traces on the fly, (2) forming context at each timepoint (*i.e.,* event), and (3) using association rules to recommend files to edit. Similar to Lee *et al.* [17] and for the sake of simplicity in the following, we use the terms "edited files" to denote the files on which edit actions occurred and "viewed files" to denote files on which selection actions occurred. Figure 1 summarises the steps of a recommendation approach where a recommendation is made at each timepoint i, *i.e.,* when the developer triggers an event e_i on a program entity.

The key basis of the recommendation approach is the context used to trigger the recommendations. For example, the context can be built by considering any combination of three viewed files within the last 100 events. Lee *et al.* [17] provides more details about interaction-based recommendation. In Figure 1,

at timepoint i, the context consists of the most recently-viewed and edited files. The context is also considered as a query that is used to trigger a recommendation.

After forming the context, association rules can be built in the database (traces that are already used for training, then for testing) to find traces that contain the viewed files V_k and the edited files E_k in their context. An association rule corresponds to a trace in the database, which is an implication in the form $Antecedent \Rightarrow Consequence$ where $Antecedent$ is a set of viewed files and $Consequence$ is a set of edited files. The edited files found in the association rules are considered candidates for recommendation. The support and confidence of the association rules are used to rank and select the association rules. The recommended candidates are ranked using their confidence. The support is the number of traces in the database containing both the antecedent and the consequence. The confidence is the support divided by the number of traces in the database containing the antecedent [32].

3 RQ1: Is there any noise in Mylyn interaction traces?

To answer this research question, we perform a quasi-experiment [35] to collect RITs and the video-captures necessary to obtain VITs. In this section, we present how we design the experiment (See Section 3.1), collect and process the data (See Section 3.2), and study noise in interaction traces (See Sections 3.3 and 3.4).

3.1 Experiment Setup

3.1.1 Subject Systems

We choose four Java open-source systems[2] because we need their source code, which the participants 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

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

Table 3: Systems, Tasks Used for the Experiment and their Descriptions

Systems	Tasks	Descriptions
ECF	irc channel output copy/select all works incorrectly	Enter IRC channel (e.g. #eclipse-dev). Right click on the chat output text area to bring up "Copy/Clear/Select All" menu. Choose "Select All". Expected: That channel's text would be selected. Actual: The root container's (irc.freenode.net) output is selected rather than the channel's output.
jEdit	Add lines number and error messages in the select line range dialog	Display next to the "Select line range:" label the number of lines contained in the file and if the user enter a wrong character (everything that's not a number) or a wrong interval you have to display the error message if the user press the "OK" button.
JHotDraw	Change the color of labels and background of the FontChooser application	The "Collections" label will be in blue, the "Family" label in yellow and the "Typeface" label in red. The background of the window will be in black.
PDE	Autostart values are not persisted correctly in the product file	On the Configuration page of the product editor, add a plug-in and set autostart to "true". Save the file. Open the file in a text editor, and see how the value of the "autostart" attribute is still set to false.

the bug reports of the two first ones while the other two have no relation with Mylyn.

3.1.2 Object Tasks

We seek concrete maintenance tasks. Thus, for each Eclipse-based system (ECF and PDE), we consider one of their bugs[3]: 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. Table 3 presents the tasks and the descriptions provided to the participants. We also provide a tutorial about the tasks.

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

3.1.3 Participants

We recruit participants via emails, which we send to the authors' research groups and individual contacts. We provide potential participants with a link[4] 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.

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 s/he will perform one maintenance task on one Java system for about 45 minutes, although there is no time limit. We also inform he/r that the collected data is anonymous. We ask the participants to try their best to complete the tasks but also tell them that they can leave the experiment at any time for any reason without penalty whatsoever.

3.2 Data Collection 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[5]. 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. All participants performed the assigned tasks without any interruption. After the participant completes the task, s/he deactivates the task to stop collecting RITs. More details regarding how Mylyn collects RITs can be found in [10].

During the tasks, we also captured video recordings of the participants' screens using VLC Media Player 2.0.5[6] at 15 images per second. We transcribed the video captures into ITs manually. The first author and two intern

[4] http://goo.gl/DyQu6j
[5] The archived systems are available in the replication package.
[6] http://www.videolan.org/vlc/releases/2.0.5.html

students from the Ptidej Team defined a transcription template after viewing the videos once to minimise subjectivity. Then, they transcribed the videos into CSV files (*i.e.,* VITs), including the following information (All the data is available online[7]):

– 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.
– 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.
– 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.
– Comments (string): Other information that the transcriber found useful for further analysis.

Table 4 shows an excerpt of the activities of participant $S09$ from the video recorded during our experiment (Column "Activities from video"), the corresponding RITs (Column "Mylyn Recorded Activities") and VITs (Column "Video-based Interaction"). For the sake of simplicity, we only show the *startDate*, *endDate*, *kind*, and *StructureHandle* for RITs and the fields relevant to time-related noise (*i.e.,* "Start" and "End") and edit-related noise (*i.e.,* "Activity") for VITs.

Table 4 shows that the participant started to perform the task at event $E1$ while the first event recorded by Mylyn is $E8$. Thus, Mylyn would miss the time between $E1$ to $E7$ (about one minute and 20 seconds). Moreover, using the activities collected by Mylyn (in Column "Mylyn Recorded Activities" in Table 4), the table shows that there are sometimes idle times and overlap times between Mylyn events (*e.g.,* idle time between $E11$ and $E14$ and overlap time between $E33$ and $E35$). These overlap and idle times would bias time analyses based on ITs. Also, some events (*e.g.,* $E33$ and $E35$) are labeled as edit activities wrongly according to the video captures.

After collecting data, we study noise at both trace and event levels. In fact, noise can be at the trace-level and–or at the event-level: at the trace-level, the

[7] http://www.ptidej.net/downloads/replications/emse15b/

Table 4: Sample of Video, the Corresponding Mylyn, and the Transcribed Activities

ID	Activities from video		Mylyn Recorded Activities				Video-based Interaction (VIT)		
	Timestamp	Description of Activities	StartDate	EndDate	kind	StructureHandle	Start	End	Activity
E1	00:04:24	Expand the project, explore the package explorer, then collapse the project					00:04:24	00:04:24	expand
							00:04:24	00:04:34	scrolling
E2	00:04:34	Expand the project, then scroll the package explorer					00:04:34	00:04:34	expand
							00:04:34	00:04:41	scrolling
E3	00:04:41	Expand then collapse the package “doclet”					00:04:41	00:04:42	expand
E4	00:04:42	Expand another package, then explore the content of the package					00:04:41	00:04:42	expand
							00:04:42	00:05:12	search
E5	00:05:12	Java search through the menu Search -> Java using the keyword "range". Then no results found					00:05:12	00:05:29	search
E6	00:05:29	Second search through the menu Search -> search using the keyword "select line range"					00:05:29	00:05:48	search
E7	00:05:48	Resize the search result view					00:05:48	00:05:56	search
E8	00:05:56	Open the file "shortcuts.xml". First interaction with the program entity (file) "shortcuts.xml" through the search result view. The keystroke is detected in the opened file in the editor.	16:40:37.618	16:40:37.618	selection	shortcuts.xml	00:05:56	00:06:01	open
			16:40:37.645	16:40:44.616	edit	shortcuts.xml			
E9	00:06:01	Expand many result entries in the search result view					00:06:01	00:06:20	search
E10	00:06:20	Open the file "jedit_gui.props" through the search result view.The keystroke is detected in the opened file in the editor.	16:41:02.409	16:41:02.409	selection	jedit_gui.props	00:06:20	00:06:20	open
			16:41:02.424	16:41:03.15	edit	jedit_gui.props			
E11	00:06:20	Resize the code editor, then read the content of the opened file					00:06:20	00:06:43	read
E12	00:06:43	Select an entry in the search result view					00:06:43	00:06:51	selection
E13	00:06:51	Search through the menu Search -> search using the keyword "range".					00:06:51	00:07:06	search
E14	00:07:06	Explore (scroll, expand and collapse many entries) the search result view					00:07:06	00:08:55	search
E15	00:08:55	Open the file "SelectionManager.java" through the search result view. The keyword is selected in the method "invertSelection()"	16:43:37.818	16:43:40.101	selection	SelectionManager.java	00:08:55	00:08:55	open
			16:43:39.141	16:43:39.141	selection	invertSelection()			
E16	00:08:55	Scroll the opened file in the editor					00:08:55	00:09:00	search
E17	00:09:00	Scroll the package explorer					00:09:00	00:09:19	search
E18	00:09:19	Scroll the opened file in the editor					00:09:19	00:09:34	search
E19	00:09:34	Expand the class to view the field and method names in the package explorer					00:09:34	00:09:53	scrolling
E20	00:09:53	Scroll the editor to read the content of the opened file					00:09:53	00:10:01	read
E21	00:10:01	Explore (scroll, expand, resize) the content of the search result view					00:10:01	00:10:28	search
E22	00:10:28	Open the file "ExtensionManager.java" through the search result view. The previewsly used keyword (range) is selected in the method "paintScreenLineRange()"	16:45:11.926	16:45:11.926	selection	ExtensionManager.java	00:10:28	00:10:39	open
			16:45:12.627	16:45:12.627	selection	paintScreenLineRange()			
E23	00:10:28	Scroll the opened file in the editor					00:10:39	00:10:55	scrolling
E24	00:10:39	Explore (scroll) the content of the search result view							
E25	00:10:55	Open the file "BufferHandler.java" through the search result view. The previewsly used keyword (range) is selected in the method "foldLevelChanged()"	16:45:37.887	16:45:37.887	selection	contentInserted()	00:10:55	00:11:00	open
			16:45:38.565	16:45:38.565	selection	foldLevelChanged()			
E26	00:10:55	Explore the content of the editor					00:11:00	00:11:14	search
E27	00:11:00	Explore the content of the search result view					00:11:14	00:11:21	search
E28	00:11:14	Search -> search using the keyword "linerange" after setting the search option for searching method. Then no results found.					00:11:21	00:11:33	search
E29	00:11:21	Search -> File using the keyword "linerange"					00:11:33	00:11:43	scrolling
E30	00:11:33	Scroll the results in the search result view							
E31	00:11:43	Open the file "SelectLineRange.java" through the search result view. The keyword is selected in the method "SelectLineRange()"	16:46:24.659	17:07:23.534	selection	ok()	00:11:43	00:12:16	open
			16:46:25.387	16:56:13.867	selection	SelectLineRange()			
E32	00:11:43	Scroll the content of the opened file in the editor					00:12:16	00:12:33	read
E33	00:12:16	Select some variables in the method "SelectLineRange()"	16:46:58.390	17:00:06.484	edit	SelectLineRange()	00:12:33	00:12:36	scrolling
E34	00:12:33	Scroll the package explorer					00:12:36	00:15:12	read
E35	00:12:36	Scroll the content of the opened file in the editor. Select some variables in the method "ok()". Then scroll and select another statement	16:47:43.450	17:08:47.639	edit	ok()	00:15:12	00:15:12	selection
E36	00:15:12	Select variables in the method "SelectLineRange()"					00:15:12	00:15:21	read
E37	00:15:21	Explore the content of the editor and try to use the contextual menu, then does not use it.					00:15:21	00:15:38	read
E38	00:15:38	Internal search in the opened file using the shortcut (the Find/replace window is open instantly). The key word "startField" is found and selected in the method "createFieldPanel()"	16:50:31.132	16:50:31.132	selection	createFieldPanel()	00:15:38	00:16:01	search
E39	00:16:01	Close the Find/Replace window and scroll the content of the file in the editor					00:16:01	00:16:18	read
E40	00:16:18	Change the method "createFieldPanel()" by adding two empty lines. Then a statement.	16:51:00.189	16:55:43.956	edit	createFieldPanel()	00:16:18	00:16:46	edit
E41	00:16:46	Scroll the content of the file in the editor					00:16:46	00:16:57	read

noise would impact the overall time spent on the task (*i.e.,* time-related noise) while at the event-level, the noise impacts the characterisation of edit events (*i.e.,* time- and edit-related noise).

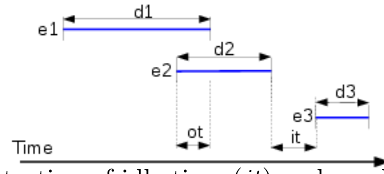


Fig. 2: Illustration of idle time (it) and overlap time (ot)

3.3 Trace Level

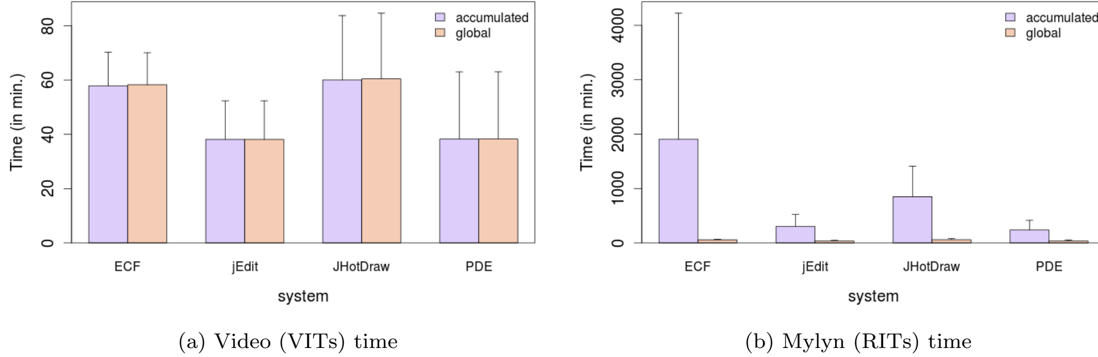
We consider and compare both VITs and RITs. At trace level, we study the possible noise introduced by the reasons discussed in Section 2.1 (*i.e.,* inability to log some events and misuse of timestamps). In fact, some researchers computed the times spent on tasks by participants considering the start and end timestamps of the whole ITs, *e.g.,* [23], while others aggregated and cleaned the times spent for each event, *e.g.,* [31].

3.3.1 Approach

We consider the times spent performing the tasks as defined below and illustrated in Figure 2, 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}). For the sake of simplicity, we use $Start(e)$ and $End(e)$ where e is an event in VITs or RITs to denote the start time and the end time of the event e. 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 2 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 2 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 is noise 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. Then, we compare the global times of VITs and RITs, and finally, the accumulated times of VITs and RITs. 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; we only want to know if there is a difference. We consider that the difference is significant at $\alpha = 0.05$ and we measure the magnitude of the difference using the Cliff's d non-parametric effect size. Following Romano *et al.* [25], we consider that the effect size is negligible if $|d| < 0.147$, small if $|d| < 0.33$, medium if $|d| < 0.474$, and large if $|d| >= 0.474$.



(a) Video (VITs) time (b) Mylyn (RITs) time

Fig. 3: Difference in global and accumulated times

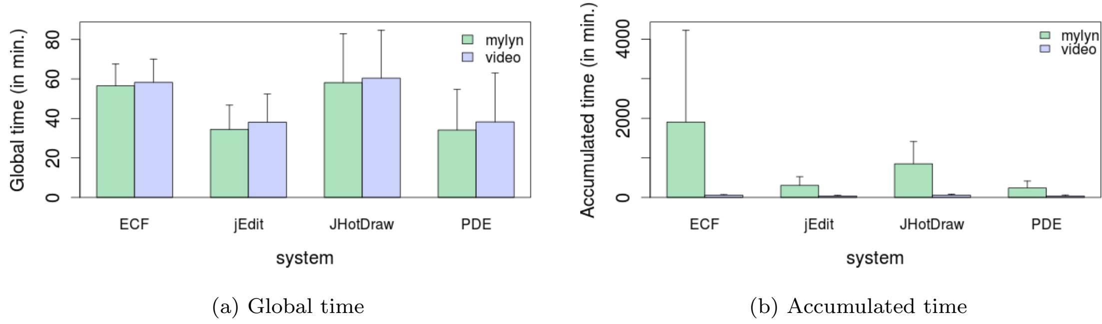
Table 5: p-values and effect size (Cliff $|d|$) of the comparison between GT and AT, and between RITs and VITs

	p-values (Cliff $\|d\|$)					
	GT vs. AT		**RITs vs. VITs**		**RITs vs. VITs**	
	VITs	**RITs**	**GT**	**AT**	**Idle**	**Overlap**
ECF	0.88 (0.12)	0.02 (0.75)	0.68 (0.25)	0.02 (0.75)	0.02 (0.75)	0.02 (0.75)
jEdit	1 (0)	0.1 (0.66)	0.4 (0.55)	0.1 (0.66)	0.06 (0.66)	0.06 (0.66)
JHotDraw	0.68 (0.25)	0.02 (0.75)	1 (0)	0.02 (0.75)	0.02 (0.75)	0.02 (0.75)
PDE	0.77 (0.18)	0.05 (0.68)	0.68 (0.25)	0.05 (0.68)	0.02 (0.75)	0.02 (0.75)

3.3.2 Results

Figure 3 presents the average GT and AT computed from VITs (see Figure 3a) and RITs (see Figure 3b). Regarding RITs, the difference between GT and AT is statistically significant for two systems (ECF and JHotDraw) and borderline for PDE system (p-value exactly 0.05), as shown in Table 5 (first part). The observed difference is due to overlap times taken into account when computing the AT. For example, while the average AT for ECF is about 1,900 minutes, the average GT for the same system is about 57 minutes. For this particular system, the large difference is due to the overlap between several selection events (on the sub-projects) that lasted about one hour each. The difference between GT and AT 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. For VITs, the difference between GT and AT is not statistically significant: both GT and AT provide the same result. Table 5 also reveals a large effect size when the difference is not statistically significant. This large effect size is due to the small size of the data-set.

Figure 4 presents the average GT (Figure 4a) and AT (Figure 4b) from VITs and RITs. Figure 4a shows that GT_{VITs} is higher than GT_{RITs}. On the contrary, AT_{RITs} is higher than AT_{VITs} in Figure 4b. We studied whether



(a) Global time (b) Accumulated time

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

the differences are statistically significant. Table 5 (second part) shows that for ECF and JHotDraw, there are statistically significant differences between AT_{VITs} and AT_{RITs}. For PDE, the difference is not statistically significant, but is 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.

3.3.3 Discussions

The comparison between GT_{VITs} vs. AT_{VITs} (see Figure 3a) that does not show the significant difference is expected by definition. In fact, when transcribing the videos into VITs, we know exactly when the participants start the task and when they end (*i.e.,* the global time is the exact time spent on the task). Moreover, the videos do not show any interruption of the work nor any overlapping of activities. Thus, the reported activities in VITs are sequential and prevent VITs from overlapping between activities (*i.e.,* AT is the time spent on the task).

The absence of significant difference between GT_{RITs} vs. GT_{VITs} (see Figure 4a) is due to the fact that GT includes idle time and removes overlap time from the time spent on the task. Indeed, in the experiment, the included idle time is the part of the time spent on the task as our participants performed the task without interruptions and the excluded overlap time must not be considered twice. The differences observed in Figure 4a, even not statistically significant, indicate a mismatch between GT_{RITs} and GT_{VITs}. We explain that this mismatch may be caused by the tool used to gather ITs and the delays between the developers' actions and the collection of the data. For example, as shown in the sample from Table 4 (column "Mylyn Recorded Activities"), we observe that Mylyn starts collecting data when the participants interact with the first program entity, *e.g.,* if a participant starts the task by scrolling without interacting with any entity, Mylyn will not collect any data (the participant started to perform the task at event $E1$ while the first event recorded

by Mylyn is $E8$). This noise does not concern only Mylyn, 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:*** *Developers may start a maintenance task by performing activities that monitoring tools may not be able to collect (e.g., scrolling, searching), which cause for example Mylyn to start logging late (first interaction with a program entity) in the task and miss on average about 6% of the times spent on the task.*

In addition to the observation above, in real-work settings, developers may interrupt their work [21]. Thus, idle times are not always the time spent on the tasks and some idle times may be real interruptions. Using Mylyn ITs, Sanchez *et al.* [27] reported that interruptions can last more than 12 minutes (some interruptions reached 8 hours). This result show that in real world situations, interruptions may occur more often than those observed in experiment. Therefore, GT (which includes idle times) may not reflect the time spent on the tasks. 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. GT seems reliable in our data-set because they come from an experiment.

The comparisons GT_{RITs} vs. AT_{RITs} (see Figure 3b) and AT_{RITs} vs. AT_{VITs} (see Figure 4b) show some differences because AT computed from RITs includes overlap times and excludes idle. Yet, according to RITs collected during our experiment, idle times are actually the part of the time spent on the task (as participants did not interrupted their work) and overlap times are considered twice. The fact that RITs contain idle and overlap times calls for the analysis of these times.

Table 5 (third part) shows that the differences between idle and overlap times in RITs and VITs are statistically significant (with large effect sizes) for three systems (ECF, JHotDraw and PDE) while it is closed to be significant (p-value = 0.06) for JEdit.

The RITs contain an average cumulative idle and overlap times of 26.08 (median 27.55) minutes and 579.3 (median = 250.4) minutes, respectively. We compute the proportion of idle and overlap times wrt. the global time by dividing the mean duration of idle and overlap times by the global time spent on the task and observe that:

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

Overall, idle times in RITs can reach more than five minutes and individual idle times lasted in average 0.5 (median = 0.07) minutes.

> ***Observation 3:*** *The collected ITs from our study reveal that idle times in ITs are not always interruptions and that the average idle time is half a minute. Moreover, our data-set shows that computing the time spent on the task required to remove overlap times from accumulated times and consider the idle times that are not interruptions as part of the time spent during maintenance task.*

We qualitatively analyse the reasons of idle and overlap times from the sample of collected RITs. These reasons and the illustration examples (from Table 4) are summarised in Table 6 and explained in the following:

- Idle times: we found three reasons for idle times. (1) Inability of Mylyn to collect some activities: Mylyn does not collect activities that do not involve a program entity such as expand/collapse a node in the package explorer or search view, resize windows, search through keywords (using shortcut or menu), scrolling; (2) Inactivity periods: sometimes developers are inactive while reading code, thinking, reading task description. These inactivity periods[8] cause idle times in RITs; (3) Breakdown of activities: Mylyn breaks down activities performed by developers. When the developers open a file for example, Mylyn generates two events corresponding to the selection of the opened file and the detection of the keystroke in the opened file. The delay from the selection of the file to the detection of the keystroke in the file sometimes causes an idle time.
- Overlap times: Overlaps between events may occur in two cases. (1) Multi selection/highlight of program entities: opening the file through the search view (after text-based search) for example causes the selection of the entities in other views (*e.g.,* package explorer, outline) and–or the highlight of the searched keywords in the opened file. In addition to the break down of activities, this behaviour sometimes causes the overlap between the generated events. The illustration corresponding to the activity $E15$ (from Table 4) is shown in Figure 5); (2) Aggregation of events: as mentioned in Section 2.1, according to the intent of the collection of ITs, Mylyn sometimes aggregates events. For example, there is overlap between $E33$ and $E34$, and $E35$. The selection of variables in the method `SelectLineRange()` (activity $E33$ in Table 4) causes an edit event. After $E34$ and $E35$, there is another selection of variables in the same program entity *i.e.,* `SelectLineRange()` (activity $E36$ in Table 4). The aggregation of $E33$ and $E36$ causes Mylyn to keep only $E33$ with the timestamps expanded from $E33$ to $E36$.

[8] The inactivity periods is practically impossible to monitor. Even being able to identify these inactivity periods from the videos, it's still hard to know what developers were doing (*e.g.,* thinking vs. reading the description of the task). However, we mainly use the detection of the mouse focus to know whether developers were reading (visual exploration) the code. Sometimes, developers were reading code when scrolling. Thus, reading code and scrolling the editor could be interchangeably and this does not affect our study regarding the kind of activity performed by the developers as the kind of activity that is mainly used in the following is the edit activity which is easy to identify.

Table 6: Reasons and Example of Activities that Cause Idle and Overlap times

	Reasons	Example (from Table 4)
Idle Times	Inability to collect some activities	$E9$, $E12$ to $E14$, $E16$ to $E21$
	Inactivity periods	$E11$
	Breakdown of activities	$E8$, $E10$, $E25$
Overlap Times	Multi selection/highlight	$E15$
	Aggregation of events	$E33$, $E36$

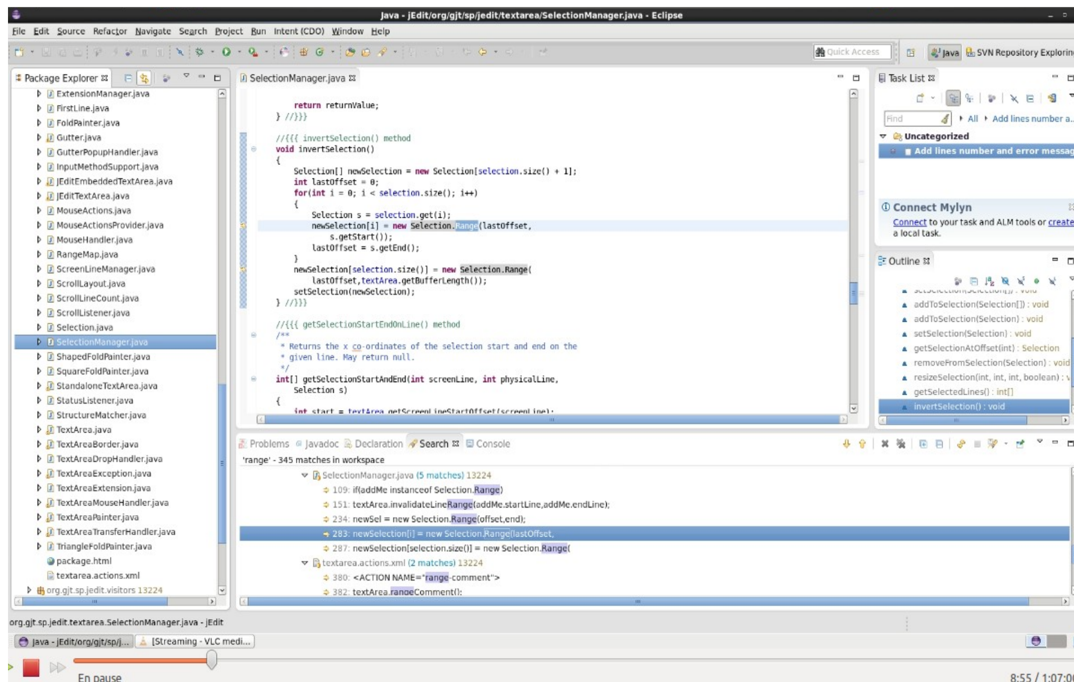
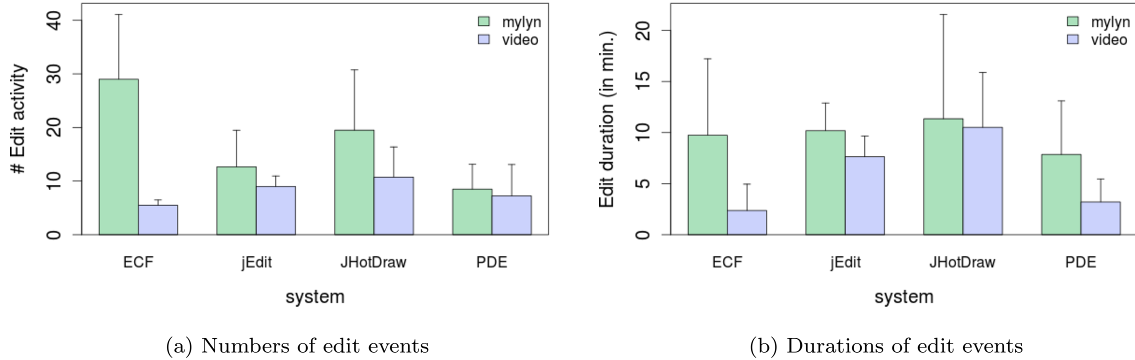


Fig. 5: Illustration of the Breakdown of Activities and the Multi-selection (Activity $E15$ from Table 4)

Beside the aggregation of events, the possibility of missing the end timestamp of some events could be another source of overlap time that should be investigated in future work.

3.4 Event Level

At the event level, we focus on the noise 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



(a) Numbers of edit events

(b) Durations of edit events

Fig. 6: Difference in edit events between RITs and VITs

of changing source code [17,27,31]. Also, edit events have been used to build code recommendation tools [9,17] and as proxy to productivity [10,27]. Thus, any noise related to edit events would impact such previous studies.

3.4.1 Approach

We consider the numbers of edit events and the times spent performing edit events to investigate noise 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 to be selection-events in previous studies [17,27].
- 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 3.3.1).

3.4.2 Results and Discussions

We compute the number of edit events and the duration of these edit events for each participant. Figure 6 shows the average number of edit events and the average duration of edit events. We observe a difference in the number of edit events between VITs and RITs (Figure 6a) and in the durations of the edit events between VITs and RITs (Figure 6b). Using the number and the duration of edit events for each participant, we compare each of them between VITs and RITs. The differences are not statistically significant as revealed by

Table 7: Comparison between T_{VITs} and T_{RITs} edit events

	Number edit		Edit duration	
	p-value	Cliff $\|d\|$	p-value	Cliff $\|d\|$
ECF	0.13	0.5	0.26	0.37
jEdit	0.8	0	0.4	0.33
JHotDraw	0.34	0.37	1	0.06
PDE	0.66	0.25	0.34	0.37

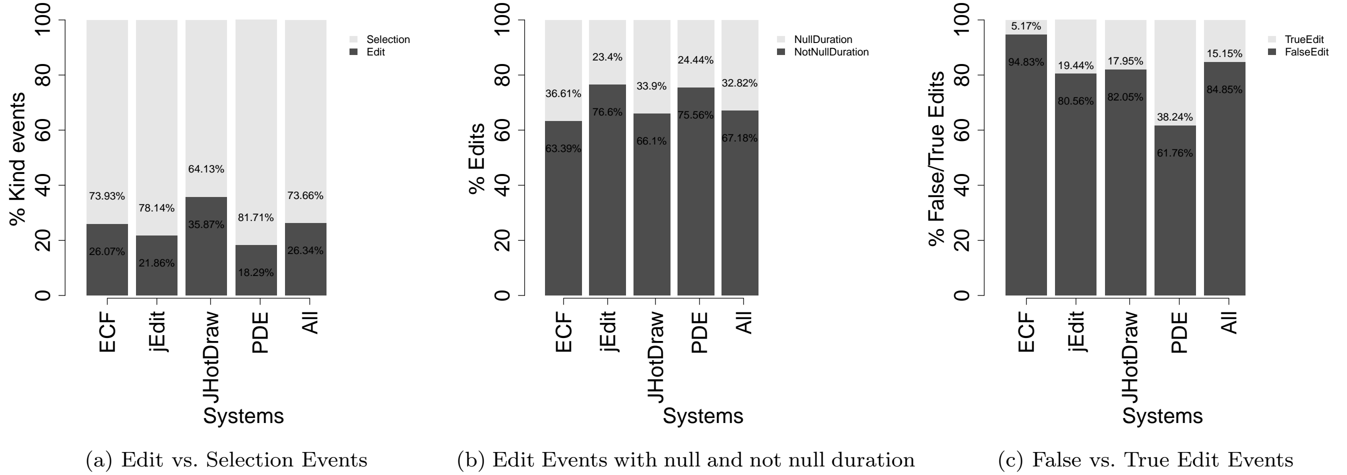
the Wilcoxon test results from Table 7. However, Table 7 shows a medium effect size because of the small size of data-set obtained from the experiment.

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. Twelve out of the 15 participants (80%) performed at least one true edit event. Among them, eight participants (53.33%) provided a patch while the other four did not provide any patch. Among the eight who provided a patch, five of them successfully completed the task: their patches fixed the bug. As resolving the task requires to change the code, results of the statistical tests would be different if all participants performed changes. Also, the absence of statistical significant difference in the numbers and durations of edit events between VITs and RITs is likely due to the low number of participants to our study and the fact that our study was not performed in a real world environment.

However, even though they are not statistically different, the observed difference in Figure 6 reveals that there are some edit events that are not real changes to the code (See Section 4.1.2 for the details about these edit events). We call these edit events "false edit events". For example in Figure 6a, the large difference in the numbers of edit events between RITs and VITs for ECF is due to the purpose of Mylyn, which directs the way Mylyn labels and aggregates events. Mylyn labels some selection of text or keystrokes detected in the editor as edit events because edit events are meant to be used to compute the relevance of a target program entity to a task. Thus, the more (false) edit events on an entity, the most it is relevant to the task, even if some of these edit events are not true edit events but related events (selection, keystrokes). Participants performing tasks on ECF performed a lot of code navigation to external libraries classes, which lead to the detection of keystrokes in the editor and resulted in false edit events.

To quantify[9] the proportion of false edit events, we manually investigate edit events in RITs by checking the video to identify what is the activity that generate each edit event in RITs. During manual identification of false edit events in RITs, in case of aggregation (*i.e.,* from the video, an edit event occurs on a program entity several times and reported once in RITs), we consider that an edit event is a false edit event if all the aggregated edit events

[9] The original version of this article [29] includes a mistake in quantifying the proportion of false edit events, which we corrected in this version. This correction affects only the proportion of the false edit events and does not affect our conclusions.



(a) Edit vs. Selection Events (b) Edit Events with null and not null duration (c) False vs. True Edit Events

Fig. 7: Proportion of the Events

are false edit events. We observe that RITs contain overall about 36% of edit events vs. 74% of selection events (See Figure 7a). About 33% of these edit events are edit events with 0-duration (See Figure 7b). Considering the edit events with non 0-duration (about 67%), 85% of them are false edit events and only 15% are true edit events (See Figure 7c). This proportion contradicts the assumption that all non 0-duration edit events correspond to changes to the code.

> ***Observation 4:*** *The comparison between the amount of edit performed by the developers and the edit events in the collected Mylyn interaction traces shows that collected ITs contain about 85% of false edit events.*

We reported our observation to the Mylyn community to have their opinion on these false edit events. The feedback we got from the Mylyn community confirms our observation. One leading developer of Mylyn stated that "*... the argument that there is noise in the edit events makes sense to me.*". They justify the prevalence of false edit events by the original premise of Mylyn edit events. "*The original premise of the edit events was that time spent in the editor is more productive than time spent looking around the structure views that eventually get you to the editor. The edit events don't have to be textual edits, which is why we decided not to base this event on something like keystrokes in the editor. For example, selecting a bunch of text, then copying and pasting it, could produce a number of edit events, and be more meaningful work than 30 keystrokes*". However, when considering selections of text as true "edit" events as suggested by the Mylyn developers, we could still find about 75% of false edit events in the studied ITs.

Overall, both time- and edit-related noise are mainly due to the monitoring tool and the kind of activity performed by the developers. The presence of noise (and in some case the significant difference between VITs and RITs) is not attributed to the variation in the participants because we observe the studied noise for all developers and tasks used in our study. The noise cannot be attributed to the task because two of the tasks used in the study were real tasks from bug repository of the systems. However, the noise discussed in this article concerns Mylyn interaction traces. We do not pretend that other tools may have the same prevalence and type of noise. In Section 8.1, we discuss other tools in comparison to Mylyn and highlight their aspects that may be subject to less/more noise. However, more studies are needed to assess if the data collected by these tools effectively contain noise and the type of noise that they may contain.

The noise observed in Mylyn ITs is relevant knowledge only if the uses of ITs with and without noise provide different results, *i.e.,* noise affects the results of studies that use ITs. Hence, we address the noise in Section 4 before studying their effect on previous work in Section 5 and 6.

4 RQ2: How can we address the noise in Mylyn interaction traces?

We now address edit-related noise, *i.e.,* false edit events. We first present a threshold-based approach [29] and then a prediction-based approach that outperforms the previous approach.

4.1 Threshold-based Approach

The threshold-based approach consists (1) to align RITs and VITs, (2) to use the alignment to identify false edit events in RITs, and (3) to define a threshold and rules to classify RITs edit events as false or true edit events.

4.1.1 Alignment

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 align the first event of a RIT with the corresponding event in the VIT using the RIT timestamp ("reference timestamps" in the remainder) corresponding to the start timestamp of the VIT.

Second, we use the reference timestamps to compute the exact time when event start and end. Let us consider an event e starting at timestamp d_1 and ending at timestamp d_2. The start and end times of e are $Start(e) = time(d_1) - time(d)$ and $End(e) = time(d_2) - time(d)$, where d is the reference timestamp and $time(x)$ is the number of milliseconds to reach x.

Once we have start and end times for all events in VITs and RITs, we can align their events if they ideally start and end at the same times. However, the

Table 8: Number of Edit and Selection Events

(a) Raw Dataset

Systems	#Selection	#Edit	Total
ECF	519	183	702
jEdit	168	47	215
JHotDraw	211	118	329
PDE	201	45	246
Total	1099	393	1492

(b) After Alignment

#False Edit	#True Edit
179	4
43	4
115	3
36	9
373	20

(c) Threshold-based Correction

#False Edit	#True Edit
179	4
44	3
116	2
37	8
376	17

Table 9: Distribution of the Time spent on Edit Events with non 0-duration

	Time (in seconds)					
	Min	**Q1**	**Median**	**Q3**	**Max**	**Mean**
False Edits	0.001	0.52	4.14	28.33	371.00	24.01
True Edits	1.14	62.6	114.4	221.8	341.2	143.7

alignment is not always ideal and 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 $End(e_1) = Start(e_2)$. Then, the alignment consists in searching the events in VITs corresponding to events in RITs. We consider that an event in a RIT 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 $\{e_1, e_2, ..., e_n\}$.

4.1.2 Identification of False Edit Events

We consider that all edit events with non 0-duration in RITs that are not aligned with any edit event in VITs are false edit events. An edit event in RITs aligned with an edit event in VITs is a true edit event. Figure 9 presents the classified true and false edit events from RITs and the time spent performing each edit events. It shows that there are few true edit events because VITs transcribed from videos are more fine-grained than RITs, leading to fewer true edit events. Some RITs events could correspond to many VITs events. For example, if a participant changes the code, scrolls the editor, performs a text-based search in the opened file, and then changes the code again, a VIT would contain two true edit events separated by one scroll event and one search event while the corresponding RIT would contain only one edit event.

Table 8a presents the numbers of selection and edit events in the RITs and Table 8b these after removing the edit events with 0-duration and aligning RITs and VITs. Table 9 shows the distribution of the times spent on false and true edit events. It reveals that false edit events with non 0-duration take on average 24.01 seconds (median = 4.14), while true edit events take on average 143.7 seconds (median = 114.4).

Table 10: Confusion Matrices for the Evaluation of False and True Edits

(a) Identification of false edit events

	Actual Label	
Classified as	**true edit**	**false edit**
true edit	TN	FN
false edit	FP	TP

(b) Identification of true edit events

	Actual Label	
Classified as	**true edit**	**false edit**
true edit	TP	FP
false edit	FN	TN

4.1.3 Threshold for False Edit Events

We must define a threshold to accurately classify false and true edit events. We assess different threshold values, then evaluate their precision and recall, and, finally, choose the most accurate threshold. We increase the threshold values per steps of ±5 seconds from the rounded mean duration of false edit events, *i.e.,* 24 seconds in Table 9. We compute precision, recall, and F-measure to evaluate the classification using the confusion matrices in Table 10.

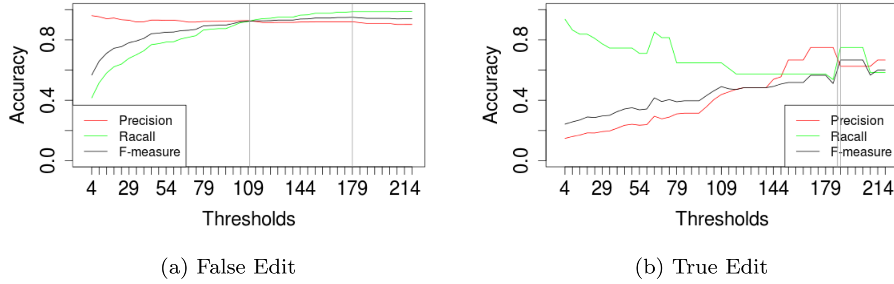
Figure 8a shows the precision, recall, and F-measure of false edit events while Figure 8b shows the precision, recall, and F-measure of true edit events. We observe that the threshold of 189 seconds maximises the F-measure of false and true edit events. However, it also results in the misclassification of most of the true edit events (gray horizontal line in Figure 9).

Yet, we want to prioritise true edit events to minimise the chance to miss them. Some true edit events may last for only a short time, *e.g.,* a refactoring. Using the mean duration of false edit events misclassifies only three true edit events (blue horizontal line in Figure 9). Thus, we consider the mean duration of false edit events as the threshold that achieves 93% precision and 64% recall for false edit events, and 18% precision and 81% recall for true edit events. 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 precision and recall of false edit events.

> ***Observation 5:*** *The average duration of false edit events is 24 seconds. Thus, we consider that an edit event is a false edit event if it takes less than 24 seconds but more than 0 second.*

Most false edit events correspond to the opening of files, the navigation between program entities, and text-based search in files. We studied RITs to distinguish which false edit event corresponds to events in the VITs. We identified one case of open event, *e.g.,* a false edit event occurring after an event triggered through the search view. Then, we built our correction approach with the annotation rules in Table 11. For example, the first row in the table means that if an event is an edit event that lasted less than or 24 seconds and if the previous event was triggered through a search view, the event should be considered an open event.

With the threshold, the precision of true edit is low compared to that of false edit events as shown in Figure 8. This relative low precision is due to (1)



(a) False Edit

(b) True Edit

Fig. 8: Precision, Recall, and F-measure of False and True Edit Events

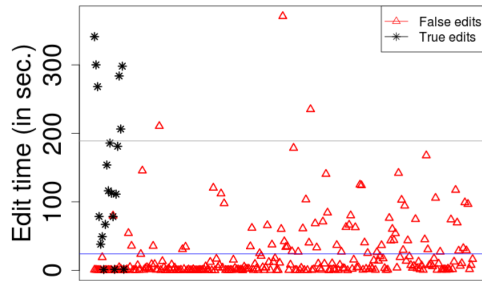


Fig. 9: Time Spent on False and True Edit Events

Table 11: Rules for annotation of edit events

RITs events	Times	Previous event	Annotated events
Edit	$time \leq 24sec$	Search view	Open
Edit	$time \leq 24sec$	Other	Navigation
Edit	$time > 24sec$	N/A	Edit

the number and kinds of false edit events and (2) the small number of true edit events. For the classification of false edit events, most of the false edit events are developers' actions that lasted few seconds, such as opening a file or selecting an identifier (names of variables, methods, classes).

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 developers performed selection events in the editor and type hierarchy view when the entities were relevant. This observation is consistent with that by Ko *et al.* [11], who

Table 12: Rules for annotation of selection events

RITs events	Origin	Annotated events
Selection	Editor	Inspection
Selection	Other	Selection

Table 13: Size of the Training and Testing Sets

	#False Edit	#True Edit	Total
Training Set	225	12	237
Testing Set	148	8	156
Total	373	20	393

stated that the selection in the editor view indicates a perception of relevance. Hence, we annotate the selection event as indicated in Table 12.

4.2 Prediction-based Approach

The threshold-based approach improves accuracy but has a low recall for true edit events. We propose a novel approach for the classification of edit events based on both supervised and unsupervised algorithms.

4.2.1 Supervised Classification

We use several classification algorithms from Weka (Naive Bayes, J48, Random Forest) to classify edit events as true or false. We first compute the following metrics that we use to classify edit events:

– Element type: The type of the element on which the event occurred.
– Origin ID: The part of the IDE on which the event was triggered.
– Interest: The degree of interest of the entity source of the event.
– Raw duration: The duration of the event, including overlaps if any.
– Clean duration: The duration of the event, after removing any overlaps.

To classify edit events, we first split the data into training and test sets. We use the random sampling method without replacement. The training set contains 60% of the data and test set 40% of the data. We obtain an imbalanced training set *i.e.,* the classification categories (false and true edit events) are not equally represented in the training set (225 = 95% of false edit events vs. 12 = 5% of true edit events). In our training set, false edit events form the largest class while true edit events are minority, which biases our classification. We avoid such bias by under sampling and over sampling the training set.

Under sampling consists in removing false edit events (majority class) from the training set while over sampling consists in adding true edit events (minority class) to the training set [8].

We apply random under sampling using the *SpreadSubSample* method in Weka, sampling majority class from 225 instances to 12 instances. We apply

Table 14: Feature Selection for Edit Prediction

Methods	Descriptions	Features
CfsSubsetEval	Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them [34]	Interest Clean duration Origin ID
ChiSquaredAttributeEval	Evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class [34]	Interest Clean duration Raw duration Element type Origin ID
FilteredAttributeEval	Evaluate the features individually through an arbitrary filter [4]	Interest Clean duration Raw duration Element type Origin ID
FilteredSubsetEval	Evaluate subset of features through an arbitrary filter [4]	Interest Clean duration

Table 15: Evaluation of the Prediction Algorithms (under sampling)

Algorithm	FS	Accuracy	Precision	Recall	F-measure
NaiveBayes	**FS1**	83.64	(98.4, 22.6)	(84, 77.8)	(90.6, 35)
	FS2	83.64	(98.4, 22.6)	(84, 77.8)	(90.6, 35)
	FS3	86.79	(98.5, 26.9)	(87.3, 77.8)	(92.6, 40)
J48	**FS1**	64.77	(99, 12.7)	(63.3, 88.9)	(72.2, 22.2)
	FS2	64.77	(99, 12.7)	(63.3, 88.9)	(72.2, 22.2)
	FS3	64.77	(99, 12.7)	(63.3, 88.9)	(72.2, 22.2)
Random Forest	**FS1**	71.69	(99.1, 15.4)	(70.7, 88.9)	(82.5, 26.2)
	FS2	71.69	(99.1, 15.4)	(70.7, 88.9)	(82.5, 26.2)
	FS3	73.58	(98.2, 14.9)	(73.3, 77.8)	(84, 25)

FS= Feature Selection approach (See Table 14)
FS1:FilteredSubsetEval, FS2:CfsSubsetEval
FS3=ChiSquaredAttributeEval or FilteredAttributeEval
(x, y) = (false edit events, true edit events)

the over sampling using the implementation of SMOTE (Synthetic Minority Over-sampling Technique) method [5] in Weka, sampling the minority class from 12 instances to 224 instances.

To evaluate the relevance of the features used to classify edit events, we apply different feature selection approaches. Table 14 shows the feature evaluation methods used and the selected features. The order of the features in Column "Features" is their order of relevance according to the selection approaches. The four approaches select "Interest" and "Clean duration" as relevant. The "Chi Square Attribute Eval" and "Filtered Attribute Eval" reveal that all the features are relevant while "Filtered Subset Eval" shows that only "Interest" and "Clean duration" are relevant.

Table 15 and 16 show the evaluation of the classification algorithms for under sampling and over sampling together with the selected features. The gray lines in each table show the better models in term of accuracy and F-measure. Table 17 shows the number of true and false edit events obtained after applying the best models of under sampling and over sampling. We can observe that NaiveBayes (using under sampling) and J48 (using over sampling) produce the best models. Moreover, these best models are obtained using all

Table 16: Evaluation of the Prediction Algorithms (over sampling)

Algorithm	FS	Accuracy	Precision	Recall	F-measure
NaiveBayes	**FS1**	84.90	(96.3, 17.4)	(87.3, 44.4)	(91.6, 25)
	FS2	84.90	(97, 20)	(86.7, 55.6)	(91.5, 29.4)
	FS3	88.05	(97.1, 25)	(90, 55.6)	(93.4, 34.5)
J48	**FS1**	83.01	(97.7, 20)	(84, 66.7)	(90.3, 30.8)
	FS2	86.79	(98.5, 29.6)	(87.3, 77.8)	(92.6, 40)
	FS3	89.93	(97.9, 31.6)	(91.3, 66.7)	(94.5, 42.9)
Random Forest	**FS1**	84.27	(96.3, 16.7)	(86.7, 44.4)	(91.2, 24.2)
	FS2	88.05	(97.1, 25)	(90, 55.6)	(93.4, 34.5)
	FS3	87.42	(97.1, 23.8)	(89.3, 55.6)	(93.1, 33.3)

FS= Feature Selection approach (See Table 14)
FS1:FilteredSubsetEval, FS2:CfsSubsetEval
FS3=ChiSquaredAttributeEval or FilteredAttributeEval
(x, y) = (false edit events, true edit events)

Table 17: Number of False and True Edit using the best J48 Over Sampling and NaiveBayes Under Sampling

	#False Edit	#True Edit	Total
J48 Over Sampling	139	17	156
NaiveBayes Under Sampling	132	24	156

the features. Considering both over and under sampling best models, J48 using over sampling provides the best results for the classification of edit events. The most important features revealed by J48 are: Interest, Clean duration, Raw duration, Element type, and Origin ID.

We evaluate the classification of false edits using 10-folds cross validation on the whole data-set. Table 18 shows that applying 10-folds cross validation using J48 and the "Filtered Subset Eval" feature selection approach provides better accuracy and F-measure (gray line in Table 18) compared to the classification with training and testing sets.

However, the resulting models (after splitting data and using 10-folds cross-validation) are not optimized. To optimize our classification model, we use the caret R package [12] for automatic parameter optimization [33]. We use the function `train` from Caret to generate the candidates parameter settings and tune the parameters using five repeated 10-folds cross validations. We define the number of levels for each tuning parameter to five [13]. To evaluate the candidates parameter settings, Caret uses all the combinations of candidates parameter settings. By default, Caret considers the parameter settings that achieve higher accuracy as the optimized parameter settings.

Table 19 shows the parameter settings for each classification algorithms (the optimal parameter settings provided by Caret is in bold). Figure 10 shows the distribution of the accuracy for the parameters. Overall, J48 performs better with the median accuracy of 95% (average 94.9%) compared to Random Forest (median accuracy = 94.8% and average accuracy = 94.8%) and Naive Bayes (median accuracy = 93.7% and average accuracy = 93.3%). Figure 11 shows that Cleaned duration, Interest, Raw duration, and Element type are

Table 18: Evaluation of the Prediction Using 10-folds Cross Validation

Algorithm	FS	Accuracy	Precision	Recall	F-measure
NaiveBayes	**FS1**	93.43	(96.2, 36.8)	(96.8, 33.3)	(96.5, 35)
	FS2	93.43	(96.3, 36.8)	(96.8, 33.3)	(96.5, 35)
	FS3	93.68	(96.3, 38.9)	(97.1, 33.3)	(96.7, 35.9)
J48	**FS1**	95.7	(96.1, 75)	(99.5, 28.6)	(97.8, 41.4)
	FS2	95.2	(95.6, 66.7)	(99.5, 19)	(97.5, 29.6)
	FS3	93.6	(94.9, 16.7)	(98.7, 04.8)	(96.7, 07.4)
Random Forest	**FS1**	93.18	(95.5, 28.6)	(97.3, 19)	(96.4, 22.9)
	FS2	93.68	(95.3, 30)	(98.1, 14.3)	(96.7, 19.4)
	FS3	94.4	(95.8, 45.5)	(98.4, 23.8)	(97.1, 31.2)

FS= Feature Selection approach (See Table 14)
FS1:FilteredSubsetEval, FS2:CfsSubsetEval
FS3=ChiSquaredAttributeEval or FilteredAttributeEval
(x, y) = (false edit events, true edit events)

Table 19: Tuning Parameters with Caret

Algorithm	Tuning Parameters	Candidate Settings
NaiveBayes	fL: Laplace Correction	**0**
	usekernel: Distribution Type	**TRUE**, FALSE
	adjust: Bandwidth Adjustment	**1**
J48	C: Confidence Threshold	0.01, 0.1325, **0.2555**, 0.3775, 0.5
	M: Minimum Instances Per Leaf	1, 2, 3, 4, **5**
Random Forest	mtry: #Randomly Selected Predictors	2, **4**, 6, 8, 11

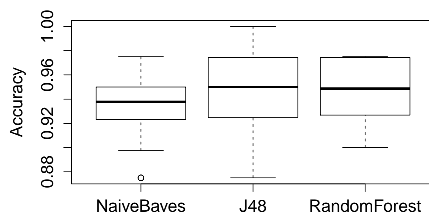


Fig. 10: Distribution of the Accuracy of Parameters tuning using Caret

the most important features to classify edit events. The final model of J48 using the optimised parameters achieves an accuracy of 96.9% for both false and true edit events, and the equal precision, recall, and F-measure of 98.4% (false edit events) and 71.4% (true edit events).

Overall, the optimized J48 model outperforms previous models and we investigate in Section 4.2.2 whether the unsupervised clustering could be an alternative to classify edit events.

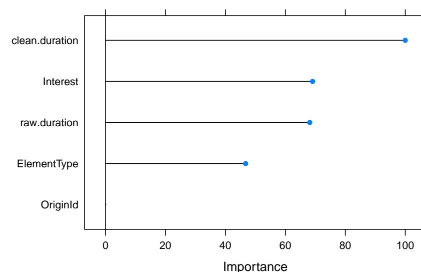


Fig. 11: The Importance of Predictors Revealed by the Parameters Tuning

4.2.2 Unsupervised Classification

We use two unsupervised classifiers for clustering edit events: K-means and spectral clustering [39]. We choose K-means because it is one of the most frequently applied unsupervised classifier and spectral clustering because it competes with supervised classifiers [39]. We applied K-means using the function `kmeans` in R. We define the number of cluster as two because we want to classify edit events as false or true. We use the R implementation of the spectral clustering provided by Zhang *et al.* [39].

For `kmeans` clustering, we use the silhouette index to evaluate the clustering. The silhouette index measures the tightness and separation of clusters [26]. The silhouette ranges from -1 to 1 and a larger silhouette index indicates a better quality of the clustering results. We use the function `silhouette` in the R package `cluster` to compute the silhouette index. The clustering achieves a silhouette index of 0.84, which reveals that the clusters are well separated.

The spectral clustering shows two clusters of size 136 and 260 instances respectively. Based on the size of the clusters of K-means (63 vs. 333) and the fact that our data-set contains less true edit events, we argue that K-means outperforms spectral clustering. Unsupervised techniques do not perform as well as the supervised technique with optimised parameters (which reach 98.4% of precision and recall for false edit events and 71.4% precision and recall for true edit events)

In conclusion, considering both supervised and unsupervised techniques for the prediction of edit events, we observe that the optimised supervised technique outperforms any other classifier. Thus, we consider the optimised J48 model in the following as the best model (*i.e.,* prediction-based approach) to predict edit events.

5 RQ3: What is the effect of the noise on editing styles and exploration strategies?

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 [36] and the study on exploration strategies by Soh *et al.* [31].

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 3. 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 two approaches described in Section 4 to obtain CITs and bCITs, respectively. Even the ITs collected from Bugzilla (*e.g.,* in realistic situation) could be different (in term of noise) to ITs collected in the experiment, we assume that if there is noise in ITs collected in the experiment, the real ITs from real situations are likely to contain noise (and maybe more because, in real situations, developers may have interruption, check their email, etc.). Thus, we apply the findings from the experiment (*i.e.,* correction approaches) to Bugzilla ITs.

5.1 Effect of Noise on the Classification of Editing Styles

We use the script[10] provided by the authors of a previous work [36] to compute our participants' editing styles. For the experiment data-set, we use precision and recall to compare the editing styles identified in RITs and CITs with the editing styles identified in VITs (considering VITs as oracle).

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

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

For Bugzilla ITs, we 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 $\rightarrow$ 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*).

For Bugzilla ITs, we compute the numbers of miscategorisation between ITs. For example, if an IT is categorised as *edit-first* using bRIT and the same IT is categorised as *edit-throughout* or *edit-last* using the corresponding bCITs, we consider that there is miscategorisation. We note "style $\rightarrow$ others" with style $\in \{\text{edit-first}, \text{edit-throughout}, \text{edit-last}\}$ to denote that there is a miscategorisation from "style" to a different style.

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

Table 20: Editing styles betwwen VITs, RITs, and CITs

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

Classified = # traces identified as the corresponding style
TP = True Positive

Table 21: 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%)

5.1.1 Results

Table 20 presents the results of identified editing styles for VITs, RITs and CITs (using the threshold-based approach) 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. The classification of editing styles from VITs shows that most of the participants (9 out of 12) follow the *edit-last* style. This result is expected because the participants were not familiar with the systems on which they worked and they spent more than half of their working time exploring the systems and performed most of the true edit events (changes) in the second half of their working time. 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 show that both RITs and CITs contains noise that affect the identification of editing styles. However, when comparing RITs with CITs, using CITs improves the precision and recall by about 4% (*edit-throughout*) and 22% (*edit-last*), respectively. The use of precision-based approach to correct ITs improves the classification of edit-first style from 0% to 50% for precision and from 0% to 100% for recall. On the contrary, the prediction-based approach decreases the precision of edit-throughout style from 22% to 14% and the recall from 100% to 50%. The precision and recall of edit-last style remains unchanged. We compare the average precision and recall of the two correction approaches (threshold-based vs. prediction-based) and observe on the experiment data-set that the prediction-based approach improves the precision by about 14% (40.74% vs. 54.76%) and recall by about 7% (44.44% vs. 61.11%).

Table 21 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 changed 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. Using the prediction-based approach on Bugzilla data-set, 55.11% of traces conserve their editing styles, while 44.88% change their editing styles. Thus, the prediction-based approach of trace correction increases by 9% the number of traces that change their editing styles.

> ***Observation 6:*** *Using the prediction-based approach to correct noise in Mylyn interaction traces reveals a misclassification of about 45% of ITs in previous study by Ying and Robillard [36].*

5.1.2 Discussions

Using our experiment data-set and the two trace correction approaches shows a small difference in the identification of editing styles. We use the categorisation technique from the original work, which is based on the percentage of edit events in the first half of an IT [36]. The considered threshold to identify editing style 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 for the threshold due to the size of the data-set, shows about 34% and 45% of changes in editing styles using the threshold- and prediction-based approach, respectively.

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. Using RITs and CITs of participant S06 shows 29% and 50% of edit events in the first half, respectively. We assumed that applying our correction approaches, which improve 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 data-set.

5.2 Effect of the Noise on Exploration Strategies

To assess the effect of noise on edit ratio and time spent used by Soh *et al.* [31] to evaluate exploration strategies, we compute the edit ratio (*i.e.,* 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 significant 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 3.3.2. 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 assess the relation between edit ratios and the times

Table 22: Comparison of Edit ratios between VITs, RITs, and CITs

	VITs vs. RITs		**VITs vs. CITs**		**RITs vs. CITs**	
	p-values	**Cliff $\|d\|$**	**p-values**	**Cliff $\|d\|$**	**p-values**	**Cliff $\|d\|$**
ECF	0.02	0.75	0.46	0.31	0.02	1
jEdit	0.1	0.66	1	0.11	0.1	1
JHotDraw	0.02	0.75	0.68	0.18	0.02	1
PDE	0.02	0.75	0.68	0.12	0.11	0.75

spent on tasks. This relation is evaluated because Soh *et al.* [31] observed that edit ratio and the time spent was related to different exploration strategies, *i.e.,* edit ratios and times spent were not correlated.

5.2.1 Results and Discussions for Edit Ratios

The analysis of the edit ratios shows significant difference between VITs and RITs, except for jEdit, in Table 22 (first column). The differences between VITs and CITs (using threshold-based approach) are not statistically significant for all systems, see Table 22 (second column). The results is the same (except JHotDraw) using prediction-based approach (p-values 0.64, 0.5, 0.02, and 0.34 for ECF, jEdit, JHotDraw, and PDE, respectively). The differences between RITs and CITs (using threshold-based approach) are system-dependent (statistically significant for two systems out of four) for the experiment data-set, in Table 22 (third column). The prediction-based approach improves this result as the difference is statistically significant for three systems out of four. Applying our approaches on 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 approaches 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 correction approaches may not fully address the noise in RITs or that other characteristics of the VITs and RITs reduce the efficiency of our approaches. Future works include studying in details this observation.

5.2.2 Results and Discussions for Times Spent

Tables 23 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 23a, first column while on RITs and bRITs, it shows positive correlations (except for the ECF system), second column of Table 23a and first column of Table 23b. After applying our correction approaches, the correlations increase for both threshold- and prediction-based approaches, 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 three out of four systems.

Table 23: Spearman coefficient between edit ratios and times spent

(a) VITs, RITs, and CITs

	Spearman correlations			
	VITs	**RITs**	**CITs (T)**	**CITs (P)**
ECF	0.31	-0.4	0.6	0.73
jEdit	-0.5	0.5	-1	0.5
JHotDraw	-0.4	0.8	1	1
PDE	0.2	0.8	1	1

(T)=Threshold-based
(P)=Prediction-based

(b) bRITs and bCITs

	Spearman correlations		
	bRITs	**bCITs (T)**	**bCITs (P)**
ECF	0.11	0.6	0.57
Mylyn	0.07	0.52	0.60
PDE	0.30	0.66	0.63
Platform	0.08	0.60	0.64

(T)=Threshold-based
(P)=Prediction-based

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 for threshold-based correction), 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.* [31] that reported an absence of correlation between edit ratio and the time spent on tasks, the result of CITs is realistic because the changes to code take more time than other activities. The observed contradiction with the result of VITs (no correlation) can be explained by the number of events in RITs. While the transcription of the videos in the VITs resulted in more events than RITs, CITs correct RITs and have the same number of edit events as VITs, although the total numbers of events are less than those of VITs. This difference in the number of events increase the edit ratios of CITs that may correlate with the time spent.

In addition to the use of ITs by researchers, ITs are mostly used on-the-fly by developers for reducing the information overload in the IDE. Our correction of noise may contribute to ease the developers' life by improving the information overload reduction through the high consideration of true edit events compared to false edit events. Thus, we investigate how noise may impact the accuracy of the recommendation of program entities (*e.g.,* filtering the package explorer to reduce information overload).

6 RQ4: What is the effect of noise on recommendation systems?

This research question complements **RQ3** by investigating the impact of false edit events on the accuracy of recommendation systems built using ITs.

6.1 Motivating Example

To exemplify the recommendation system based on ITs by Lee *et al.* [17], let us consider two ITs belonging to two maintenance tasks T_1 and T_2 where T_1 is completed and the developer is performing T_2. Figure 12 shows the sequence of files involved in T_1 and T_2. The files in the circle are those that the developer edited. Thus, the context of T_1 is (V_1, E_1) with $V_1 = \{a, b, c, f\}$

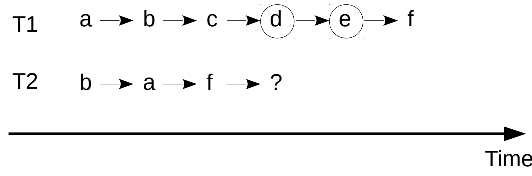


Fig. 12: Example of Recommendation

and $E_1 = \{d, e\}$. When mining T_2, the set of viewed files in the context of T_1 is used to identify the file to recommend.

Now, let us assume that the files to edit to resolve T_2 are $E'_2 = \{h, e, d\}$. When the developer performing T_2 interacts with f, the current context is (V_2, E_2) with $V_2 = \{b, a, f\}$ and $E_2 = \varnothing$. The recommendation systems uses the context and the previous association rule, whose antecedent contains the context, *e.g.,* $V_2 \subset V_1$, to recommend $E_1 = E_2 = \{d, e\} \subset E'_2$.

As explained in Section 6.4, the precision and recall at the timepoint when the developer edits f in T_2 are respectively:

$\frac{|Recommended \cap Expected}{|Recommended|} = \frac{|\{d,e\} \cap \{h,e,d\}|}{|\{d,e\}|} = 1$ and

$\frac{|Recommended \cap Expected}{|Expected|} = \frac{|\{d,e\} \cap \{h,e,d\}|}{|\{h,e,d\}|} = 2/3$

The precision and recall are computed at each timepoint and averaged by the number of recommendations. The fact that some edit events may be false edit events impacts the precision and recall of the recommendation.

6.2 Approach

To answer **RQ4**, we replicate the approach described in Section 2.3, which we use as *baseline approach*. We correct the false edit events using the threshold-based approach described in Section 4.1 and the prediction-based approach proposed in Section 4.2 and consider both as the *approaches with correction*. For a fair comparison between the baseline approach and the approaches with correction, we use the default parameters values of the baseline approach. We set the maximum number of recommended files to 10 and the support and confidence of the association rules to 1 and 0.1, respectively.

The baseline approach orders the traces using both the bug IDs for which the traces were collected and the trace IDs. This ordering does not reflect the order in which the traces were collected by developers. Therefore, we order the traces using the start timestamps of the first events in the traces. The events are by definition ordered by their start timestamps. Thus, we ensure that the recommendations are made in the chronological order of the events.

Table 24: Subject Systems

	#T	**Baseline**		**Correction**		Difference
		$\frac{\#V}{\#T}$	$\frac{\#E}{\#T}$	$\frac{\#V}{\#T}$	$\frac{\#E}{\#T}$	
Mylyn	2,726	14.39	2.75	14.40	1.64	1.10 (40%)
Platform	582	24.23	2.67	24.23	1.55	1.12 (42%)
PDE	536	6.36	1.07	6.36	0.45	0.62 (58%)
ECF	308	9.30	0.72	9.30	0.33	0.39 (54%)
MDT	245	54.48	0.71	54.48	0.39	0.32 (44%)
Others (67)	1,367	22.81	1.41	22.82	0.78	0.63 (44%)

#T = number of Traces
#V = number of viewed files
#E = number of edited files

6.3 Subjects Systems

We use the Mylyn ITs collected and shared by the developers involved in Eclipse projects and provided by Lee *et al.* [17]. The data-set involves 72 projects and contains 5,764 interaction traces. Mylyn, Platform, PDE, ECF, and MDT are the systems with most interaction traces while the 67 other projects have an average of 20 interaction traces.

Table 24 shows the number of traces (#T), the number of viewed files per trace ($\frac{\#V}{\#T}$), and the number of edited files ($\frac{\#E}{\#T}$). For example, for Mylyn, there are 2,726 traces in which, on average, 14 files are viewed and 3 files are edited per trace while, after correction, 14 files are still viewed and 2 files are edited per trace. Column "Difference" indicates the numbers and percentages of false edit events, *e.g.,* 40% of the files per trace in Mylyn were not edited.

6.4 Dependent and Independent Variables

We use the following variables to assess the impact of the corrections of false edit events on the accuracy of the recommendations. As *independent variable*, we have the considered approach: either the *baseline approach* or the *approaches with correction*. As **dependent variables**, we evaluate the recommendations using the set A of recommended files and the set E of expected files at each timepoint in the traces and using the following measures:

- Precision: $P = \frac{|A \cap E|}{|A|}$. The higher the precision, the lower is the number of incorrectly recommended files.
- Recall: $R = \frac{|A \cap E|}{|E|}$. The higher the recall, the higher the number of edited files that are recommended
- F-measure: F-measure $= \frac{2*P*R}{P+R}$. The higher the F-measure, the more accurate are the recommendations.
- Feedback: $Fb = \frac{\#Recommendations}{\#Query}$ [40]. The higher the feedback, the more efficient is the recommendation system with respect to the number of queries [17].

Table 25: Recommendations Accuracy for the Baseline Approach

	Baseline Approach				
	P	**R**	**Fb**	**#Rs**	**#Qs**
Mylyn	0.69	0.55	0.24	8,920	36,563
Platform	0.88	0.39	0.22	2,722	12,020
PDE	0.49	0.73	0.07	176	2,240
ECF	0.30	0.72	0.04	43	966
MDT	1.0	0.29	0.09	194	2,041
Others	0.77	0.36	0.11	1,817	16,356

Table 26: Recommendations Accuracy for the Correction Approaches

(a) Threshold-based Approach

	P	**R**	**Fb**	**#Rs**	**#Qs**
Mylyn	0.75 ↑	0.64 ↑	0.32 ↑	8,104	25,156
Platform	0.87 ↓	0.54 ↑	0.35 ↑	2,861	8,099
PDE	0.78 ↑	0.71 ↓	0.15 ↑	196	1,271
ECF	0.81 ↑	0.95 ↑	0.11 ↑	80	705
MDT	1.0 →	0.62 ↑	0.12 ↑	211	1,742
Others	0.79 ↑	0.60 ↑	0.16 ↑	1,559	9,441

(b) Prediction-based Approach

	P	**R**	**Fb**	**#Rs**	**#Qs**
Mylyn	0.79 ↑	0.72 ↑	0.30 ↑	5,341	17,594
Platform	0.83 ↓	0.66 ↑	0.33 ↑	2,132	6,451
PDE	0.96 ↑	0.84 ↑	0.08 ↑	81	968
ECF	0.86 ↑	1 ↑	0.05 ↑	23	423
MDT	1.0 →	0.91 ↑	0.11 ↑	121	1,019
Others	0.93 ↑	0.72 ↑	0.16 ↑	1,114	6,948

Table 27: Differences in Accuracy (F-measure) between the Baseline and Correction Approaches

	Threshold-based		**Prediction-based**	
	p-value	**Cliff** $\lvert d \rvert$	**p-value**	**Cliff** $\lvert d \rvert$
Mylyn	< 2.2e-16	0.15	< 2.2e-16	0.31
Platform	< 2.2e-16	0.40	< 2.2e-16	0.66
PDE	0.30	0.05	0.001	0.19
ECF	1.8e-12	0.74	2.8e-10	0.93
MDT	< 2.2e-16	0.99	< 2.2e-16	1
Others	< 2.2e-16	0.49	< 2.2e-16	0.66

6.5 Analysis Method

To assess the impact of the correction approaches, we first simulate the recommendations without any corrections and compute the values of the dependent variables. Then, we correct the traces, simulate the recommendations again, and recompute the variables. Finally, we compare the three sets of values using the non-parametric Mann-Whitney Wilcoxon test with a 95% confidence level. If differences exist, we measure their magnitudes using the Cliff's d non-parametric effect size described in Section 3.3.1).

6.6 Results and Discussions

Table 25 shows the results obtained with the baseline approach including precision (P), recall (R), Feedback (Fb), numbers of recommendations (#Rs) and query (#Qs). Table 26a shows the results obtained after applying the threshold-based correction approach. We observe that the precision and recall increased after correction, except for the precision of Platform (which

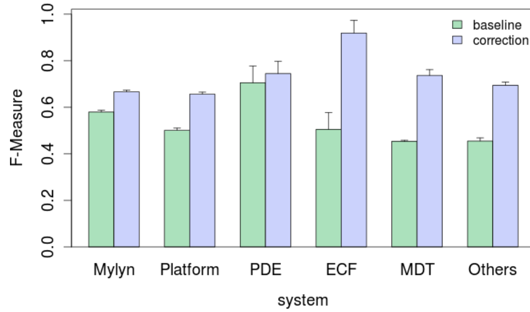


Fig. 13: Difference in F-Measure Between the Baseline Approach and Threshold-based Approach

decreased by 1%), the recall of PDE (which decreased by 2%), and the precision of MDT (which remained constant at 100%). Precision and recall for Mylyn increased from 69% to 75% and from 55% to 64%, respectively. Table 26b shows the results after applying the prediction-based approach: the recall increases by 29% when compared to the threshold-based approach.

We apply Mann-Whitney Wilcoxon test on the F-measure because it combines and represents the tradeoff between precision and recall. The test confirms that the differences are statistically significant, except for PDE using the threshold-based approach, as shown in Table 27. Figure 13 illustrates the magnitude of the differences in F-measure. The threshold-based approach yields large effect sizes for ECF, MDT, and Other systems, medium for Platform, and small Mylyn. The prediction-based approach yields the same trends as the threshold-based approach. From these results, we conclude that correcting false edit actions in interaction traces data can improve the accuracy of recommendation systems.

Observation 7: *Correcting noise in Mylyn interaction traces with the prediction-based approach can improve the precision and recall of recommendation systems by up to 56% and 62%, respectively.*

Beside the increase in accuracy, after corrections, the recommendation system provides more recommendations than with the baseline approach. For example for Platform, the baseline approach provides about 22 recommendations out of 100 queries (Feedback = 0.22 in Table 25) while after corrections, there are 35 and 33 recommendations out of 100 queries, respectively for threshold- and prediction-based approach (Feedback = 0.35 and 0.33). Thus, the corrections approaches decrease the numbers of queries and–or increase the number of recommendations. They allow the recommendation system to make more recommendations with respect to the number of queries.

We used the parameter values used by Lee *et al.* [17]. However, tuning the parameters may further increase accuracy. We varied the maximum numbers

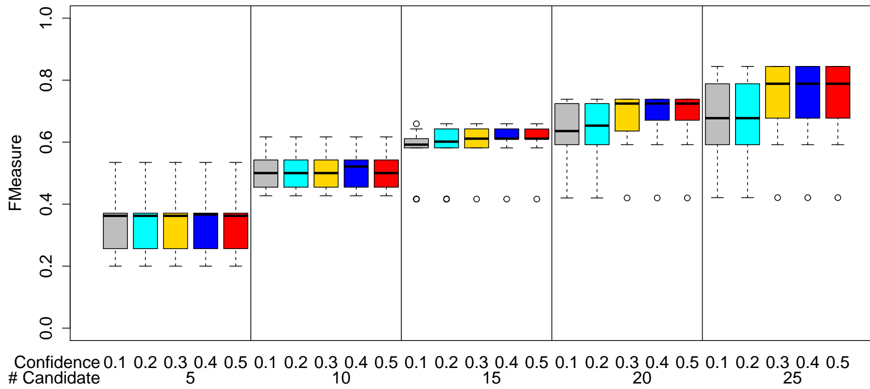


Fig. 14: F-Measure of the combination of Confidence and the number of Candidate for Support = 1.

of recommended candidates from 5 to 25 per step of 5, the support from 1 to 9 per step of 2, and the confidence from 0.1 to 0.5 per step of 0.1. We used the F-Measure to evaluate the combinations of parameters values. Figure 14 shows the distribution of the accuracy for each combination of confidence and maximum number of candidates for support equal to one. Other values of support yielded empty recommendations that we excluded. We observed that the higher the number of candidates, the higher the accuracy. When the number of candidates is 5, the variation of the confidence does not change the accuracy of the recommendations. When the number of candidates is 25, confidence values of 0.3, 0.4, and 0.5 provide high accuracy.

Table 28 shows the precision and recall of the recommendations using 25, 1, and 0.5 as number of candidates, support, and confidence, respectively. Both threshold- and prediction-based correction approaches yield improved accuracy, except for Platform as with the default parameters values in Tables 26a and 26b. For MDT, accuracy is 100% because the MDT system has many more viewed files than other systems as shown in Table 24. Thus, each association rule has a large antecedent with a large number of viewed files and it is always possible to find a rule which antecedent contains the current context.

In conclusion, the best parameter combination does not compensate for the noise in ITs. Correcting noise provides more accurate recommendations.

7 Threats to Validity

This section discusses the threats to the validity of our study.

Table 28: Precision and Recall of the Recommendation using the better Parameters (Maximum Candidate = 25, Support = 1, Confidence = 0.5).

	Baseline		**Threshold-based**		**Prediction-based**	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Mylyn	0.68	0.67	0.75 ↑	0.71 ↑	0.79 ↑	0.77 ↑
Platform	0.88	0.70	0.79 ↓	0.78 ↑	0.77 ↓	0.83 ↑
PDE	0.49	0.73	0.79 ↑	0.71 ↓	0.96 ↑	0.84 ↑
ECF	0.27	0.90	0.81 ↑	0.96 ↑	0.86 ↑	1 ↑
MDT	1	0.73	1 →	1 ↑	1 →	1 ↑
Others	0.77	0.59	0.79 ↑	0.77 ↑	0.93 ↑	0.79 ↑

7.1 Interaction Traces

We used interaction traces collected by the Mylyn Eclipse plugin [20]. Thus, the studied noise, the correction approaches, and the impact of the noise on previous work only applied to Mylyn ITs. The same noise may be more or less prevalent in the ITs collected by other monitoring tools. Section 8.1 summarises the difference between Mylyn and other tools and discusses the possibility for less/more noise in the data collected by other tools. Yet, more studies should be performed with other tools to assess the presence of the same or other noises and their impact in their collected data.

Another threat when using Mylyn was the version of the tool. During our experiment, we used different versions of Eclipse and, consequently, different versions of Mylyn, for different tasks. We were constrained in our experiment to use specific versions of the Eclipse-based systems (*i.e.,* ECF, PDE) because we used the versions against which the bugs were reported. Using different versions meant that ITs could be different. However, as features are not significantly different between versions of Mylyn, the differences between the ITs of two versions of Mylyn were negligible.

The overlap phenomenon could also affect our results. We considered the overlap between two consecutive events but not among several events, which could affect our results and is future work.

7.2 Transcription of the Video

The first author and two intern students in the Ptidej Team independently translated the videos into ITs. Because some actions were difficult to interpret and because the exact start and end timestamps of some actions were difficult to identify, different transcriptions of a video may provide different results. To mitigate this threat, we defined a common template for video transcription. Moreover, the noise studied in this article are related to timestamps and edit events, which are easy to identify on video captures.

7.3 Alignment of Mylyn and Video ITs

To identify the mismatch between RITs and VITs, we aligned both Mylyn and video captures ITs. The alignment consisted of identifying the events corresponding to one another in the two sets of ITs. As the alignment was based on the times when the events occurred, any shift may result in the lost of alignment between two corresponding events. To mitigate this threat, we manually checked some alignments and observed that false edit events, for example, were not due to shifts in alignments.

We also evaluated the accuracy of our alignment approach by manually classifying edit events as true or false edit events. We computed the precision, recall, and F-measure of the alignment. Our alignment approach achieved 91% to 100% precision and 91% to 98% recall for false edit events. To identify true edit events, our alignment approach achieved 25% to 75% precision and 14% to 100% recall. The accuracy of our alignment of false edit events supported the limited impact of the alignment on our results.

7.4 Correction of ITs

We corrected ITs using two correction approaches. The accuracy of the threshold-based approach depends on the alignment and timestamps of false edit events. The timestamps may be different for other systems and–or other developers (*e.g.,* developers' experience and speed when triggering events). We introduced the prediction-based approach to mitigate this threat from the threshold-based approach and observed that the two approaches provided the same trends. Our correction approaches may also result in corrected traces without any edit events, which may happen, for example, if developers performed refactorings in the last seconds of an IT or explored source code without making changes.

Other approaches to improve the use of ITs exist that combine ITs with other data source, such as change history. Although the combination of ITs with other data source could mitigate the impact of noise in ITs, it could also make the prediction less accurate: it could increase the level of noise. Future work must assess the benefits of our noise correction approaches over approaches that combined ITs with other data source.

To quantify the benefit of our correction approaches on recommendation systems, we used the data-set with and without correction and compared the accuracy values of the resulting recommendations using the same recommendation techniques. While the corrected and uncorrected data-sets may seem different, the only difference is the correction applied to the raw data-set. Hence, any difference in the recommendation accuracy values between uncorrected and corrected data-sets was due to the correction of the data.

7.5 Participants, Task, and Systems

Participants performed maintenance tasks on Java systems only. Hence, we cannot guarantee that using other systems and other programming languages would yield the same results. We chose the tasks to last about 45 mins. We chose tasks that may not be representative of tasks performed by developers in industrial settings.

We cannot guarantee that the 15 participants found all possible source of noises in ITs. In addition, the small number of true edit events in the data-sets could affect the performance of our prediction-based approach. Hence, a study that uses larger data-sets could observe and report a different prevalence and–or kind of noises and result in different performance of the prediction-based approach of trace correction.

7.6 Reliability

We provided all the data online. We defined a video transcription template and three people transcribed the videos. Thus, the transcriptions by other people should not be significantly different from ours (*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.

8 Related Work

While some previous studies focused on the analysis of ITs to understand developers' behaviour, others used ITs to study software engineering activities. Some other studies built tools to support developers in their daily work. These three uses of ITs are complementary and want to empower developers.

8.1 Monitoring Developers' Interactions

In addition to Mylyn, other tools exist to monitor and collect ITs. These tools differ in their goals, the collected data, and the supported IDE.

Mimec is an Eclipse plugin to collect and use ITs to improve fault notification [15]. Mimec extends Mylyn to collect more fine-grained data concerning edit and navigation activities but exclude scrolling events. Mimec data is essentially similar to that of Mylyn [14, p.23]. Thus, Mimec may suffer from the same type of edit-related noise than Mylyn as reported in Table 2. However, without excluding scrolling events may reduce event-related noise.

Blaze is an extension of Visual Studio to collect ITs including names and types of events, file names and selected lines in the files, and users' unique

IDs [7]. Blaze collects search and navigation events but not edit events. Thus, it is not suitable to study edit-related noise and time-related noise.

FeedBaG is another extension of Visual Studio to collect ITs including event timestamps, selection and edit events. FeedBaG aggregates edit events on the same program entity when the time between edit events is less than 2 seconds and the duration of the resulting event equals the time between the first and the last aggregated events [1]. This aggregation may cause less time-related noise than in Mylyn.

DFlow is an extension of Pharo to collect ITs including the start time of the events, the program entities, and the editing session start and end times [18]. It distinguishes between exploration, inspection, and editing events. DFlow includes a model to estimate the duration of edit events, which may lead to time-related noise.

WatchDog is an Eclipse plugin that monitors developers' testing activities. It groups coherent events to form intervals containing the type and the start and end times of the events [3]. Types includes read, modify, and execute a test. It does not provide individual events with start and end times.

Other monitoring tools may contain less or different edit- and time-related noise. Our study shows the need to study noise in these tools in future work.

8.2 Uses of Interaction Traces to Understand Developers' Behaviour

Ying and Robillard [36] studied developers' editing styles as edit-first, edit-last, and edit-throughout. They observed that enhancement tasks are associated with edit-first. Zhang *et al.* [38] studied developers concurrently editing a file and identified concurrent, parallel, extended, and interrupted editing patterns. Editing patterns impact software quality because they relate to future bugs.

Soh *et al.* [31] 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 visited). Unreferenced explorations require more effort but are less time consuming than referenced ones. Murphy *et al.* [19] studied Eclipse views and perspectives and reported that developers use the views differently, *e.g.,* none used the declaration view.

8.3 Uses of Interaction Traces to Study Software Engineering Activities

Sanchez *et al.* [27] related work fragmentation (due to interruptions) and developers' productivity: work fragmentation correlated with low productivity. Robbes and Röthlisberger [23] correlated developers' expertise and their effort on their tasks: a negative correlation between the time spent on a task and the developer's expertise. Soh *et al.* [30] correlated the complexity of task resolutions (*i.e.,* using submitted patches) and the time spent on the tasks: effort and implementation complexity are not correlated. Bantelay *et al.* [2] combined ITs with commit data to identify evolutionary couplings: combining

ITs and commit data improved the recall of prediction models by up to 13%, with a decrease in precision of less than 2%. Zanjani *et al.* [37] improved a change-impact analysis using ITs.

8.4 Uses of Interaction Traces for Recommendation Purposes

Tools exist to build the context of a task [10] and reduce the developers' information space. Yet, developers still must look into the reduced information space to find relevant program entities. Recommendation systems exist to provide developers with this relevant information in a given context [24]. They include TeamTrack [6], which uses the association between previously-visited program entities to recommend the next entity, NavTrack [28], which uses selection- and open-actions on files to discover hidden dependencies between files and recommend next files, NavClus [16], which improves previous systems using navigation information, and MI [17], which includes view/selection histories of program entities to recommend entities that have not yet been edited.

Kersten and Murphy [10] used interaction traces to build task contexts and recommend program entities 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 [22] also used change history to reduces developers' scrolling effort and to complete code. Zimmermann *et al.* [40] mined the version histories and used co-change relations among entities to recommend entities to be changed.

Instead of studying developers' behaviour and activities, our study investigated noise in ITs and assessed its impact on developers' editing styles and exploration strategies and on recommendation systems. Our study relates to the works above because it provides insights to improve tools based on ITs.

9 Conclusion and Future Work

Developers' interaction traces (ITs) have been used for several purposes, *e.g.,* assessing how developers' editing stules [38] and recommending program entities [17]. However, ITs are subject to several assumptions: edit events caused by changes to the code and the times mined from ITs reflect the times spent by developers on their tasks. Yet, ITs collected by different tools and in non-controlled environments may be subject to noise.

In this article, we studied the noise in ITs collected by Mylyn, an Eclipse plugin, and the impact of this noise on the studies of developers' editing styles and the accuracy of recommendation systems. We conducted a quasi-experiment and asked 15 participants to perform four maintenance tasks on Java-based systems. We collected both ITs and video captures of the participants' screens. We compared the collected ITs (RITs) and video captures (VITs) and observed that Mylyn ITs miss up to 6% of the times spent performing tasks and that they contain about 85% of false edit events.

Then, we proposed to predict false edit events using a threshold- and a prediction-based approach. The prediction-based approach outperforms our previous threshold-based approach, achieving up to 98% precision and 98% recall. We performed an experiment using ITs collected from Eclipse Bugzilla issue tracker system. We reported that noise may have led researchers to mislabel some participants' editing styles for about 45% of the ITs. Correcting false edit events can significantly improve the accuracy of the recommendations of program entities by up to 56% for precision and up to 62% for recall.

We thus recommend that researchers interested in building new recommendation systems based on Mylyn ITs and developers using these recommendation systems to guide their maintenance activities correct existing traces data prior to their usage. Our results are specific to Mylyn interaction traces and similar studies should be performed on data collected by other monitoring tools (*e.g.,* Mimec, FeedBaG, DFlow) to assess whether they contain noise, the types of noise, and their prevalence, and the sources of the noise.

Future work includes leveraging our knowledge of noise in ITs to improve monitoring tools, such as Mylyn. One of Mylyn developers suggested that we clean the identified noise during the creation of events as we "*can capture more information at that point, for example, whether there was a keystroke performed that correlates to the edit selection*". We also plan to extend our study to include more complex tasks and investigate other possible noise in ITs (*e.g.,* noise in the program entities involved in the ITs), as well as using corrected ITs to assess developers' expertises.

Acknowledgment

The authors greatly thanks the participants of the experiments described in this article. This study would not have been possible without their participation. Many thanks also go to intern students Thomas Drioul and Pierre-Antoine Rappe, who transcribed the video captures. Thanks to Seonah Lee and colleagues for providing us with the data and the implementation of their recommendation system. We also thank Annie Ying and Martin Robillard for making their script publicly available for use. The authors are grateful to the editors and anonymous reviewers for their detailed feedback and useful suggestions to improve the earlier version of this article. This work has been partially funded by a NSERC Discovery grant and the Canada Research Chairs on Multi-language Patterns and on Software Change and Evolution.

References

1. Amann, S., Proksch, S., Nadi, S.: Feedbag: An interaction tracker for visual studio. In: 24th IEEE International Conference on Program Comprehension, ICPC 2016, pp. 1–3 (2016)
2. Bantelay, F., Zanjani, M., Kagdi, H.: Comparing and combining evolutionary couplings from interactions and commits. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, pp. 311–320 (2013)

3. Beller, M., Gousios, G., Panichella, A., Zaidman, A.: When, how, and why developers (do not) test in their ides. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 179–190 (2015)
4. Bouckaert, R.R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., Scuse, D.: WEKA Manual for Version 3-7-8 (2013)
5. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. **16**(1), 321–357 (2002)
6. DeLine, R., Czerwinski, M., Robertson, G.: Easing program comprehension by sharing navigation data. In: Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on, pp. 241–248 (2005)
7. Fritz, T., Shepherd, D.C., Kevic, K., Snipes, W., Bräunlich, C.: Developers' code context models for change tasks. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, pp. 7–18 (2014)
8. Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T., Matsumoto, K.i.: The effects of over and under sampling on fault-prone module detection. In: Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on, pp. 196–204 (2007)
9. Kersten, M., Murphy, G.C.: Mylar: a degree-of-interest model for ides. In: Proceedings of the 4th International Conference on Aspect-oriented Software Development, AOSD '05, pp. 159–168 (2005)
10. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT/FSE, pp. 1–11 (2006)
11. Ko, A., Myers, B., Coblenz, M., Aung, H.: 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 (2006)
12. Kuhn, M.: caret: Classification and regression training. URL `https://cran.r-project.org/web/packages/caret/index.html`
13. Kuhn, M.: Building predictive models in r using the caret package. Journal of Statistical Software **28**(5), 1–26 (2008)
14. Layman, L.M.: Information needs of developers for program comprehension during software maintenance tasks. Ph.D. thesis, North Carolina State University (2009)
15. Layman, L.M., Williams, L.A., St. Amant, R.: 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, pp. 73–76 (2008)
16. Lee, S., Kang, S.: Clustering navigation sequences to create contexts for guiding code navigation. Journal of Systems and Software (2013)
17. Lee, S., Kang, S., Kim, S., Staats, M.: The impact of view histories on edit recommendations. Software Engineering, IEEE Transactions on **41**(3), 314–330 (2015)
18. Minelli, R., Mocci, A., Lanza, M., Kobayashi, T.: Quantifying program comprehension with interaction data. In: 14th International Conference on Quality Software, QSIC 2014 (2014)
19. Murphy, G.C., Kersten, M., Findlater, L.: How are java software developers using the eclipse IDE? IEEE Software **23**(4), 76–83 (2006)
20. Mylyn: http://eclipse.org/mylyn/
21. Parnin, C., Rugaber, S.: Resumption strategies for interrupted programming tasks. Software Quality Journal **19**(1), 5–34 (2011)
22. Robbes, R., Lanza, M.: Improving code completion with program history. Automated Software Engineering **17**(2), 181–212 (2010)
23. Robbes, R., Röthlisberger, D.: Using developer interaction data to compare expertise metrics. In: Proceedings MSR, pp. 297–300 (2013)
24. Robillard, M., Walker, R., Zimmermann, T.: Recommendation systems for software engineering. Software, IEEE **27**(4), 80–86 (2010)
25. Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J.: Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? Annual meeting of the Florida Association of Institutional Research (2006)
26. Rousseeuw, P.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**(1), 53–65 (1987)

27. Sanchez, H., Robbes, R., Gonzalez, V.M.: An empirical study of work fragmentation in software evolution tasks. In: Proceedings SANER, pp. 251–260 (2015)
28. Singer, J., Elves, R., Storey, M.A.: Navtracks: Supporting naviga-tion in software maintenance. In: International Conf. on Software Maintenance, pp. 325–334 (2005)
29. Soh, Z., Drioul, T., Rappe, P.A., Khomh, F., Gueheneuc, Y.G., Habra, N.: Noises in interaction traces data and their impact on previous research studies. In: Empirical Software Engineering and Measurement, 9th Int. Symposium on (2015). To appear
30. Soh, Z., Khomh, F., Gueheneuc, Y.G., Antoniol, G.: Towards understanding how developers spend their effort during maintenance activities. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, pp. 152–161 (2013)
31. Soh, Z., Khomh, F., Gueheneuc, Y.G., Antoniol, G., Adams, B.: On the effect of program exploration on maintenance tasks. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, pp. 391–400 (2013)
32. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, chap. 6: Association Analysis: Basic Concepts and Algorithms. Pearson (2006)
33. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 321–332 (2016)
34. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc. (2005)
35. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering - An Introduction. Kluwer Academic Publishers (2000)
36. Ying, A., Robillard, M.: The influence of the task on programmer behaviour. In: Proceedings ICPC, pp. 31–40 (2011)
37. Zanjani, M.B., Swartzendruber, G., Kagdi, H.: Impact analysis of change requests on source code based on interaction and commit histories. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 162–171 (2014)
38. Zhang, F., Khomh, F., Zou, Y., Hassan, A.E.: An empirical study of the effect of file editing patterns on software quality. In: Proceedings WCRE, pp. 456–465 (2012)
39. Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E.: Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, pp. 309–320 (2016)
40. Zimmermann, T., Weiβgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. IEEE Transactions on Software Engineering **31**(6), 429–445 (2005)