JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2016; **00**:1–32
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Work Fragmentation in Developer Interaction Data

Heider Sanchez[‡], Romain Robbes[‡], Luis C. Cruz[§], Victor M. Gonzalez[§]

[‡]*: Computer Science Department (DCC), University of Chile, Chile;* [§]*: Instituto Tecnologico Autonomo de Mexico, Mexico*

Received . . .

1. INTRODUCTION

Work fragmentation is a phenomenon that has been extensively investigated in the literature. Several observational studies in company settings have shown that work fragmentation is very common in the workplace. These studies have also shown that work fragmentation is detrimental to the actual work taking place: after such a context switch, time is necessary for information workers to regain their bearings. A particularly harmful kind of work fragmentation is interruptions, where an external signal (email, chat, phone call, or direct conversation) forces an information worker to switch activity at an unplanned moment and for an unknown duration.

If the literature contains extensive studies of interaction workers, it lacks in two aspects: (1) most studies target information workers (fewer target the specific population of software developers), and (2) the studies are usually field studies, which are limited in the amount of data they contain.

Work fragmentation is indeed an important phenomenon within the context of modern software development, and the impact there may be even worse since developers build and maintain complex mental models of the software they are working on—these models may be more sensible to interruptions, and costly to rebuild.

In this paper, we present a study addressing these issues by (1) being focused on software developers, and (2) using MSR techniques to base its conclusions on a much larger amount of data (specifically, thousands of development sessions recorded by Mylyn and the Eclipse Usage Data Collector). The downside is that the data exploited, being operational data [1], is of lower quality than data extracted from a full-fledged observational study. Trading quality for quantity allows us to explore different aspects of the problem, and to generate hypotheses for subsequent studies of work fragmentation and interruptions in software development. We address the following three questions:

- **RQ1:** What is the relationship between the observed interruptions and the observed developer productivity? Every interruption requires a time to regain the lost context, so in this research question we intend to see an accumulative effect when the number of interruption increases.
- **RQ2:** Is the observed relationship more pronounced in the presence of prolonged interruptions? A longer interruption might be more detrimental to the productivity than several small interruptions, as it requires more time to recover from it; this research question explores that possibility.

*Correspondence to: Romain Robbes (rrobbes@dcc.uchile.cl)
†All the authors are supported by LACCIR Project RFP2012LAC004
H. Sanchez is supported by a research grant from CONICYT-Chile
R. Robbes is supported by FONDECYT project #1151195

- **RQ3:** What is the observed relationship in the vicinity of interruptions? Based on the literature and observational studies, we believe that in the immediate time around an interruption the effect of it can be more palpable.

This work is an extension to a published paper in the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering. This improved version includes the same analysis previously made with a new dataset, the Eclipse Usage Data Collector (UDC) interaction data, which allows us to dig further into more specific aspects, resulting in the following two complementary research questions:

- **RQ4:** How is the developer's performance during the recovery time after an interruption? Our interest is to analyze the evolution of the productivity as the programmer recovers from an interruption.
- **RQ5:** What events are more common during recovery time? The pattern of activities during the recovery time might change in comparison with other phases; in this research question we investigate if there is such a difference.

A note on terminology: in this paper we (ab)use the word *interruption* to refer to a *gap of activity* observed in IDE activity. We do not presume that all of these activity gaps are actual interruptions (our results lead us to believe otherwise); rather, we assume they are indicators of work fragmentation in general (that is, a programmer switching contexts to perform other activities while developing, such as answering email, looking for the solution of a problem on the Internet, etc). There also actions taking place outside of the computer, such as taking phone calls, interacting with people or leaving one's desk. Of note, work by Zou and Godfrey [2] found that interruptions were a common cause for a lack of observed activity, and classified all periods of inactivity as interruptions.

Structure of the paper. We start with a literature review of studies of work fragmentation in Section 2. This section also contains a review of studies performed with the Mylyn and UDC datasets that we use. In Section 3, we describe the Mylyn dataset in further detail, highlighting potential issues in the operational data that we use in the analysis. In Section 4, we describe the UDC datasets and its particularities. In Section 5, we answer our first research question: we find an inverse relationship between number of interruptions and our three productivity indicators. In Section 6, we answer our second research question, finding that this relationship is more pronounced for development sessions with at least one longer interruption. In Section 7 we answer our third research question and find that our productivity indicators are indeed lower in the vicinity of interruptions; we further find three patterns of interruptions with different characteristics. In Section 8, we answer the fourth research question, finding a gradual increase of productivity during the recovery time after an interruption and a relation with the nature of the interruption, whether is positive or negative. Finally Section 9 answers our fifth research question; based on the type of events executed, we find differences in the activities performed after a positive or negative interruption. In Section 10, we discus about the differences between both datasets. We close the paper by discussing the threats to validity of this study in Section 11, before concluding in Section 12.

2. RELATED WORK

2.1. Empirical Studies of Work Fragmentation

Studies of information workers In the modern workplace, people routinely multitask and shift their attention to multiple areas, projects and activities. An observational study by Gonzalez and Mark found that information workers experience high level of multitasking, averaging 3 minutes on a task before switching to another task [3]. This often results in work fragmentation and interruptions which some studies found has detrimental effects to knowledge workers, such as stress and frustration [4]. Work fragmentation as result of interruptions usually demands extra effort to recover and resume pending activities: a study of 24 information workers found that a worker needs

DOI: 10.1002/smr

on average 25 minutes to get back on an interrupted task [5]. Similarly, Iqbal and Horvitz [6] found that people experience disorientation and loss of context when multitasking. Czerwinsky et al. found that after experiencing work fragmentation people found it more difficult to perform interrupted tasks and took longer to complete them [7].

There is also evidence that work fragmentation can be beneficial to some extent. For example, Aral et al. [8] indicate that multitasking is associated with more project output to a point, after which the productivity decreases. Adler et al. [9] also notice a breaking point of the effect of multitasking on productivity, describing it as an inverted-U relationship, where low or high levels of multitasking is detrimental, but there is a middle ground where the programmer takes advantage of the state of alertness.

While these studies already show the effect of work fragmentation can be hurtful, they are mostly observational studies over a limited time, and do not address specifically the sub-population of programmers. According to Parnin and Rugaber [10], programmers must perform activities that challenge the memory, like keep track of refactoring changes, navigate unfamiliar code, learn a new API and understand new concepts; when this tasks are interrupted in can be costly to recover the lost state. Some of this tasks require to invest in program comprehension activities, which can account from 54 to 95% of the time spent on every working session [11].

Studies specific to programmers Zou and Godfrey performed an analysis of interaction histories of 3 industrial programmers for a month [2]. One of their findings was the omnipresence of inactivity in the interaction data, that they interpreted as interruptions. They corroborated this with the programmers which agreed that many such periods of inactivity were indeed interruptions.

Similar to other information workers, software developers experience work fragmentation due to the nature of the activity. Ko et al. [12] performed an observational study of 17 Microsoft developers, where the average time before a switch was five minutes. Some switches were due to necessary changes between tasks, but others were due to interruptions.

Parning and Rugaber evidenced the presence of an edit lag in the majority of a large sample of development session spanning several datasets, hence showing that developers need time to resume work after an interruption [13]. Parnin and DeLine evaluated several cues to help programmers resume work after interruptions forcing them to multitask [14].

Maalej et al. [15] performed a study on the program comprehension strategies of software developers. They found that developers often preferred direct interaction to consulting documentation. As such, experts on a piece of code were often interrupted in their work to answer questions from others.

Fritz et al. studied the perception developers have of their productivity. They found that developers perceived they were more productive on days where they accomplished significant tasks, and when they were not significantly interrupted [16].

As above, most of these studies operate on small datasets. The study of Parnin and Rugaber is the closest to ours in terms of amount of data considered. However, the analysis performed on large datasets in their study focused principally on the edit lag metric, while we investigate metrics which are indicators of productivity along the entire session.

2.2. Empirical Studies on Mylyn Data

Several studies mining interaction data have been presented. Kersten and Murphy evaluated the effectiveness of their degree of interest (DOI) model by showing that in a field study of developers, users of their tools had a significantly higher edit to selection ratio [17]. Fritz et al. evaluated how well DOI values reflected the developers knowledge of the code, and encountered mixed results [18]. The lessons they learned allowed them to develop a Degree of Knowledge (DOK) model, an extension of the DOI containing authorship information, and that was found useful in a variety of situations [19]. The DOI has been implemented in Mylyn, and the data it generated has been used in several studies. Based on the data generated by an early version of Mylyn (which recorded additional data, namely usage of commands), Murphy et al. described how 41 Java developers used Eclipse, reporting on the most commonly used views and commands [20]

DOI: 10.1002/smr

The Mylyn data available in the Eclipse issue tracking system was used in several studies. Ying and Robillard characterized the edit strategies of developers in the Mylyn dataset, finding three: edit-first, edit-last, and edit-throughout [21]. Lee et al. developed Micro Interaction Metrics in order to enhance defect prediction models [22]. We investigated whether the duration of a task could be used to evaluate the accuracy of expertise metrics, with promising early results [23]. Soh et al. studied the exploration patterns in the developer sessions of the Mylyn datasets, and classified their exploration strategies as referenced or unreferenced exploration; they find that unreferenced exploration were less time consuming [24]. Soh et al. conducted another study of this dataset, finding that the effort spent by a developer in a patch is not correlated with the implementation complexity of the patch [25]. Bantelay et al. improved the accuracy of evolutionary couplings metrics for change prediction (usually computed with commit data); by adding interaction data, recall increased by 13%, with a 2% drop in precision [26]. Zanjani et al. integrated this approach with information retrieval techniques to support impact analysis based on textual descriptions of change requests [27].

2.3. Empirical Studies with Usage Data Collector

The Usage Data Collector dataset is a large collection of information about interaction data of Eclipse's users. It is publicly available and has been used in several studies mining interaction data.

On refactoring activities, Murphy-Hill et al. used this data to understand how programmers are using refactoring tools analyzing patterns on refactoring practices, finding that most programmers don't make deep use of these tools, leaving untouched most of the configuration parameters and performing manually most of the refactoring [28]. Liu et al. identified refactoring tactics (floss refactoring and root canal refactoring, following Murphy-Hill's terminology), finding that the majority of them are floss refactoring and detecting differences on activities performed depending on the type of the refactoring [29]. Sahin et al. analyzed the energy usage changes after refactoring and used the UDC data to identify the most common refactoring events and selected those that fulfill the requirements of being common and cause structural changes [30].

On coding tools and strategies, Murphy-Hill et al. investigated command recommendations for software developers depending on their past history of command usage and tested a set of algorithms using UDC data to obtain CVS and editing commands, getting positive feedback on a live evaluation with users [31]. Yoon et al. use UDC data to compare the results obtained from a proposed plug-in for the capture of low-level events, finding a very similar report on command distribution from both tools [32].

And on a different note, Khodabandelou used the UDC data to prove a proposed model for mining user's intentions and strategies from recorder user's traces, resulting on a model for UDC developers activities and the construction of a map followed by them [33].

3. MYLYN DATA DESCRIPTION

3.1. Mylyn Data

For our analysis, we used the Mylyn dataset of development data. Mylyn [17] is an Eclipe plugin that monitors the program elements a programmer interacts with in order to build a task context. A subset of Eclipse developers (principally from the Mylyn and PDE Eclipse projects) use the Mylyn Monitor tool to capture fine-grained usage data of their IDE that they attach to the bug fixes as a task context they submit to Eclipse. This allows reviewers of the bug fixes to use the same task context when they review the changes. The task context contains the entire interaction history since a developer activated the task he or she was working on, and as such is a rather reliable account of IDE usage over time (barring a few issues explained below).

To collect the data, we crawled the bugzilla data of the Eclipse project (`http://bugs.eclipse.org`), and downloaded all the bugs that had as an attachment a Mylyn task context. In total, there were 6,182 bug reports which contained 8,102 Mylyn task contexts.

DOI: 10.1002/smr

3.2. Interaction History Format

The interaction history is a sequence of ordered events in time [21]. An event is associated to a direct action of the programmer in program elements, for instance: edit and selection events. Other interaction events are indirect [17]: they are issued by Mylyn itself while it is maintaining its DOI model of a programmer's task context. However these events are not edit or selection events. Each event captures several pieces of information: the timestamp, the kind of event and the signature of the code element that was interacted with (package, class, attribute, or method signature—including name and parameters).

In Table I we show the different kinds of events and their description; in Table II we show an example of an interaction history. However, certain characteristics in the data present challenges around the data mining we plan to perform [34]. These cases must be detected and resolved in order to get a representative time series model of programmer activity. We describe these issues and our solutions below.

Table I. Kinds of interaction events [17] in Mylyn.

kind	mode	description
selection	direct	Editor and view selections via mouse
edit	direct	Textual and graphical edits
command	direct	Operations such as saving, building, preference setting
manipulation	direct	Direct manipulation of interest
propagation	indirect	Interaction propagates to structurally related elements
prediction	indirect	Capture of potential future interaction events

Table II. Example of interaction history.

	StartTime	EndTime	EventKind	Method
1	10:30:00	-	selection	m1
2	10:30:40	-	manipulation	-
3	10:30:40	-	edit	m1
4	10:31:03.700	-	selection	m1
5	10:31:03.800	-	selection	m2
6	10:31:03.850	-	selection	m3
7	10:32:05	10:33:07	edit (5)	m2
8	10:33:10	-	prediction	-

3.3. Special Characteristics of Mylyn Data

Below, we describe the characteristics of Mylyn data one has to consider before processing them. This is especially relevant in our case since our study needs a representation of the activity as close to reality as possible. In Section 3.5 we describe the criteria used to process these characteristics.

Aggregate events. This type of event includes several actions on the same program element. These actions usually occur within an interval of short duration time. Whenever an aggregation occurs, the event is expanded to include two timestamps defining a range of time, instead of a single timestamp, and a number of events. Accordingly, these aggregate events lose their specific time besides the range of time. The reason of this is that for scalability—in terms of storage—Mylyn does not register all the user events.

For example, Table II shows an aggregate edit event in row 7. We indicate in parenthesis the number of actions associated and the field *EndTime* registers the timestamp of the last occurrence. Clearly, too much aggregation in a given trace severely compromises the detection of work fragmentation as this relies on accurate timestamps.

DOI: 10.1002/smr

Massive events. Such an event occurs when the same action is executed on more than one program element in a very short time. This massive action generates consecutive events of the same kind and with a tiny gap between them.

For example, massive events are produced when we select an entire group of classes from the navigation tree panel in Eclipse. In Table II we show a massive selection on the methods m1, m2 and m3 in rows 4, 5 and 6. This massive selection produces three consecutive events with a time gap of not more than 0.1 seconds. These events overstate the activity of developers as each of these does not correspond to an individual developer action; rather, the entire sequence is.

RomainPerhaps another example here: it was not clear for one of the reviewer why this was necessary. E.g.: if I scroll down in the outline view with the keyboard I will be generating many selection events. Check other characteristics too (agregate, long).

Very long events. These events have a duration time ($|EndTime - StartTime|$) much larger than the mean. This mainly occurs in aggregate events. We believe that this issue is due to factors related to Mylyn, since it can register the end time of an event when the task is resumed after a long downtime. Another cause of these events is when one selects a code fragment and maintains this action for a long time.

After exploring a sample of traces, we noticed that a long interruption of activity implicitly splits a trace into two sub-traces. Moreover, we have realized that many long events happen around this border. That is, they began before the interruption and finished immediately after the interruption, which confirms the observation above (resumption of a task after a long downtime). However this has the side effect of hiding the gaps of activity in the sequence of events if care is not taken. Consequently, the duration of these events is overstated; however, finding their actual duration is not trivial.

3.4. Selection

We used the phases of the process *Knowledge Discovery in Databases (KDD [35])* to discover useful knowledge from our collection of data: we first select data, then preprocess it, before transforming it to time series.

Our purpose is the analysis of work fragmentation and interruptions in software development. Therefore, we focused only in the interaction data of the user with the program elements. The variables of interest for our study are [23]:

1. Duration: the time that a programmer took to perform a programming task.
2. Edition: the amount of code changes that were necessary to perform said task.
3. Navigation: how many program elements were consulted in a task; represented by selection events.
4. Edit ratio: as in previous studies [17], the ratio of edits over edits and selections is an indicator of more efficient work since program exploration is reduced.

First, we have kept only the edit and selection events that are associated to a program element. These type of events are distributed in 8058 traces. On the other hand, edit-type events also occur when the programmer double clicks on a file. In these events the starting time and ending time are the same [22]. We consider that the double-clicking actions form part of the user navigation, therefore these events are transformed into selection events.

Second, we have kept all traces without aggregate information. Unfortunately, we cannot know the distribution in time of the actions that have been aggregated in a single event: without additional information, an aggregation of 15 events over one hour is as likely every 4 minutes as it is to have 10 events in the first 5 minutes and 5 more in the last 10 minutes. Therefore, designing a correct disaggregation task to this traces is very difficult, if not impossible.

In order to keep as many traces as possible, we have also kept a little group of traces with aggregate information where the aggregate events have a maximum duration time of five minutes (to minimize uncertainty) or have only two aggregate actions (since the actual event timestamps are known in this case). Thus, we were left with 6260 traces.

DOI: 10.1002/smr

It could be possible that these traces we discarded were biased in one way or another. In Figure 1, we note that in our data the traces with aggregation information appear from the year 2008 on, and slowly increase in frequency from then. We would expect such a pattern from a change to the version of Mylyn, rather than a change to the type of tasks being performed. This is corroborated with our tests of a recent version of Mylyn, which seems to be even more aggressive in discarding information (it only keeps information at the class level, discarding method level information). As such, our assumption is that it is unlikely that this change is due to anything else than the version of Mylyn that was in use (and hence is not due to different tasks, people, etc). This leads us to believe that it is unlikely that filtering these traces would introduce other sources of bias.

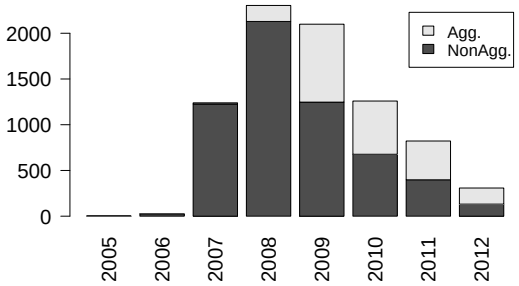


Figure 1. Grouping by date all the traces, with aggregate information (light gray) and without aggregate information (dark gray).

Finally, we found that 26% of traces had at least an interruption over 8 hours, which is hence large enough to represent the difference between two working days. In these cases, we treated these long activity gaps as splitting points in order to decompose a long trace in several development sessions [36].

After this, we considered a minimum duration time of 30 minutes in order to ensure a minimum of activity during a session. In this way, we obtained a final total of 4284 useful sessions to be processed in the next phase.

3.5. Preprocessing

After filtering out traces, we describe the final processing steps that yield the final development *sessions* that we study:

Sorting. We sort chronologically all the events by their starting time. Before that, we normalize the timezone of each trace. We found 156 traces (3%) with more than one timezone.

Massive events. We use a short spacing interval to join consecutive massive events in only two, the first and the last event (Figure 2). We have considered a spacing size of 0.1 seconds (100 milliseconds). All the massive events we found were of type "selection", which concurs which our observation above (multiple selection of several entities at the same time). The total number of selection events was reduced by 45%.

Disaggregation. We disaggregate all the actions associated with each remaining aggregate event with equidistant separation, as was done by Ying and Robillard [21]. Due to the filtering above, this was only applied on traces that had aggregate events of short duration ($\leq$ 5 minutes).

Event splitting. Finally, we split all the long duration events to better fit a time series model, which is composed of a series of actions in specific times with intervals of the same duration. A normal event does not generally have duration (EndTime is null). However, 12% of normal events had a duration > 0 seconds and 3% ≥ 1 hour, these events are outliers. Therefore, we split each long event in two events: one at the start and the other at the end of the interval. This is the same criteria applied for massive and aggregate events.

Table III shows the ratio of variation of the number of events after preprocessing.

DOI: 10.1002/smr

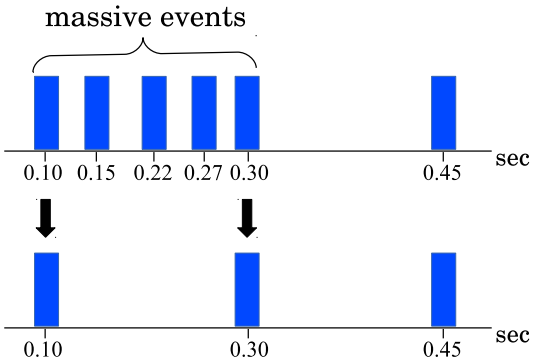


Figure 2. Replacing five massive events with only two events.

3.6. Transformation

Our goal is to build compact and representative models from each session. In this sense, we used aggregation of events to generate a multivariate time series (MTS). An MTS is a sequence of multivariate observations taken at continuous time intervals coming from a same phenomenon. We build the MTS with edit and navigation variables; the time unit is the minute, and every observation is the sum of all the events that occurred in this minute. We selected the minute as unit time because it seemed to be an appropriate and minimal representation of the user interaction in a programming task—obviously a subjective decision.

Creating time series is a useful way of representing temporal data because they allow us to apply clustering techniques and find recurrent patterns; in our case every minute of the time series contains different observation, which is the reason why we created multivariate time series. This way we can model a session as a MTS, where every minute contains the quantification of edition and selection events. The partition of the data into sessions do not alter the results since we are interested in the values by minute, so having longer sessions do not necessarily increase the number of events per minute.

Mylyn data has another characteristic which is unusual for time series: the time of occurrence of the events is not periodic (see column *StartTime* in Table II). That is to say, events occur with non-equidistant separation gaps between them. However, in a time series, the values must be evenly spaced and chronologically sorted. For this reason, we compressed the size of the multivariate time series, pulling apart all the interruptions as a new time series variable. Then, each time series value represents the interruption duration in this minute (Figure 3).

Consequently, the temporal component represents the *real working time* of a programming task, excluding inactivity. This allows us to compute our activity indicators (number of edits, selections, edit ratio) independently of the amount of inactivity in a session.

Defining an interruption. An important task during the transformation phase is to identify the interruptions along a working session. In this study we use two kind of interruptions, and we define them as follows:

Table III. Number of events after preprocessing.

type event	before	after	% variation
edit	452,236	475,554	+5%
selection	658,768	527,652	-20%
total	1,111,004	1,003,206	-10%

DOI: 10.1002/smr

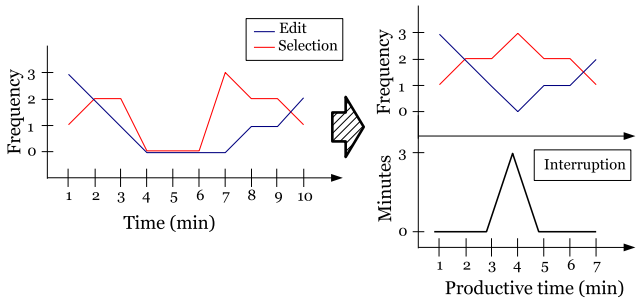


Figure 3. Example of how to compress a time series.

- *Interruption.* We define empirically an interruption as a pause of programming of duration ≥ 3 minutes. This is based on previous work where we observed that short interruptions lasted usually this long [3].
- *Prolonged interruption.* Based on additional observations from this work, we defined a prolonged interruption as one lasting for more than 12 minutes. These thresholds are also supported by the Activity Theory models of Kaptelinin and Nardi [37]. This study presented work fragmentation at two different levels: actions and activities. Interruptions originated after a period of around three minutes of sustained attention to the previous action were considered when people were switching at the level of interactions with artifacts of people. Interruptions originated after a period of twelve minutes of sustained attention to a previous activity where considered when people were switching at the level of interactions with projects or topics.

We identified that 98% of sessions had at least one gap of activity. Moreover, we observe that the short interruptions predominate over the prolonged interruptions (Table IV). This first result tells us that work fragmentation is extremely prevalent in our dataset.

Table IV. Prevalence of interruptions according to their duration time.

duration	ratio	examples
$[3-12min\rangle$	69%	short pause, answer a question, thinking, looking for interruption
$[12-30min\rangle$	18%	coffee break, short meeting, extended interruption
$[30min-2hr\rangle$	9%	lunch break, a meeting
$[2hr-8hr\rangle$	4%	extended meeting

After the transformation phase, we had the sessions in a proper format to carry out the analysis. In the Figure 4 we can observe some statistics (left) about the sessions in Mylyn and the distribution (right) of the productive time. The productive time of the session is the total time without considering the time consumed by interruptions. Also, the outliers are ignored by setting a threshold of three times the standard deviation. This filter is used only to show the distribution and avoid a very long tail provoked by unusually long sessions.

4. ECLIPSE USAGE DATA COLLECTOR DATA DESCRIPTION

4.1. The Eclipse UDC

The Usage Data Collector (UDC) dataset is a large compendium of information about interaction data from users of Eclipse, collected from December 2008 to August 2010, with the intention to keep track of how programmers are using the IDE. The framework listens to the events triggered by the user or the system, such as: edition and navigation commands; the startup of a plug in; or

DOI: 10.1002/smr

stat	value
total	4,284
min.	2 min
1st Qu.	18 min
median	35 min
mean	48.91 min
3rd Qu.	62 min
max.	535 min

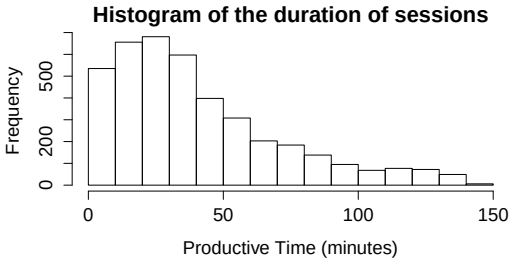


Figure 4. Statistics (left) and distribution of the productive time (right) of the sessions in Mylyn

the closing of the platform. To be more specific, UDC collects information about loaded bundles, commands accessed via keyboard shortcuts, actions invoked via menu or tool-bars, perspective changes, view usage and editor usage. The UDC is a large dataset that contains information of around 1,800,000 users, and has a total of 2,323,233,101 unique rows with 5 attributes each. Table V shows a description of the attributes.

Table V. Attributes in UDC data.

attribute	description
userId	Unique number that identifies a user
what	The action of the event (deactivated, activated, opened, executed, etc.)
kind	What kind of event was executed (workbench, view, command, etc.)
bundleId	Description of the event's package
bundleVersion	Version of the bundle's event
description	Description of the event
datetime	Date and time in UNIX format

4.2. Selection

We used the pre-processed version of the data that is published on Google BigQuery, by Murphy-Hill et al. [38]. This is an alternative version of the original UDC dataset, which is cleaned and preprocessed, so that the transformation phase is simple and focused on our needs. Due to the magnitude of the dataset we only worked on a fragment of it. We took a random sample of 1,000 users that is larger than the dataset we were able to obtain for the Mylyn study.

The first step was to query the dataset to get the data from 1,000 random users. We delimited the query to obtain only those events dispatched by the user, ignoring system events. We also ignored the bundle version and transformed the UNIX date into a more legible datetime format. From this query we extracted 4,321,349 unique events, which are around 0.18% of the whole dataset.

4.3. Preprocessing

After obtaining the data, the next step was to add the attributes required by every event in order to match as closely as possible the format of the Mylyn data; this allowed us to reuse a part of our analysis. Hence, we added fields corresponding to the type of the event (edition or selection), its duration (time elapsing between one event and the next one, used to determine where interruptions take place and sessions end) and an ID to identify the different working sessions present on the data. For the latter, we sorted the data by *userId* and *datetime*. This was required because, by default, the user's data is mixed and we need it not only chronologically correct but also sorted by users to tag the working sessions of every user without interferences.

It was also required to fill the description for the events that indicate the activation or deactivation of the workbench of Eclipse, but as they were the only events with this error this task was trivial. Additionally, it was needed to classify the events, which was a complicated task.

DOI: 10.1002/smr

4.4. Classification of events

Mylyn events belong to two main categories: edition and selection. The UDC data doesn't have this, so we inferred a classification manually. Note that we reuse the same term of selection in the following text, as it was used in Mylyn data. Strictly speaking, the word "navigation" would be better suited to the variety of events present in the UDC data, but we keep the old term for consistency.

We took into consideration the attributes *what*, *kind* and *description* to choose the best classification for every event. Starting by the kind attribute, some events are labeled as *view* which are precisely those that open, activate or close a view such as the console, the package explorer or the variables view. Therefore, the events of the kind *view* were classified as selection-type events.

The classification for the rest of the events is more complex: we needed to check directly the command description and the bundle it belongs to in order to precisely infer its meaning (e.g., We could label as edition all those commands with the attributes *command* and *executed*). The majority of them are in fact of that type edition, but some executed commands also can open a view, navigate through the text or go to another class. For this reason we took a closer look into the bundle name and what the description can tell us about the events.

Thus, by default, all events of the attributes *command* and *executed* were classified as edition and afterwards we did a thorough analysis to identify those that are actually selection events. This way we categorized as selection all the events where we could tell (based on the description) if they have the purpose to, for instance, navigate to another window, open a perspective, find and replace words, search code, debugging, or perform window management. The description contains the whole path from the top package to the class that implements the event, so we look for key words to apply the classification. The proportion of events after the classification is shown in the Table VI. We used an iterative process involving two of the authors of the paper to arrive to this result; the process was iterative, until we reached consensus. A sample of the classification can be seen in the Table VII.

Table VI. Proportion of events.

event	percentage
edition	34.6%
selection	31.1%
system	31.5%
control	0.8%

Table VII. Sample of classified events in UDC.

description	type
org.eclipse.ui.edit.delete	edition
org.eclipse.ui.file.save	edition
org.eclipse.ui.edit.undo	edition
org.eclipse.jdt.ui.CompilationUnitEditor	selection
org.eclipse.ui.console.ConsoleView	selection
org.eclipse.debug.ui.commands.StepOver	selection
org.eclipse.ui.workbench	system
org.eclipse.team.sync.views.SynchronizeView	control

The edition and selection types are the ones we are interested for this study. The system classification is composed of the events that activate and deactivate the workspace of Eclipse; it is difficult to determine when and why these events are executed (they can be executed either via user or system action, and the order of execution in relation with associated events is not clear), so we decided to establish this particular classification for them. Finally, the events of the type control are those related to the version control system.

DOI: 10.1002/smr

stat	value
total	15,825
min.	2 min
1st Qu.	11 min
median	23 min
mean	35.87 min
3rd Qu.	47 min
max.	1,345 min

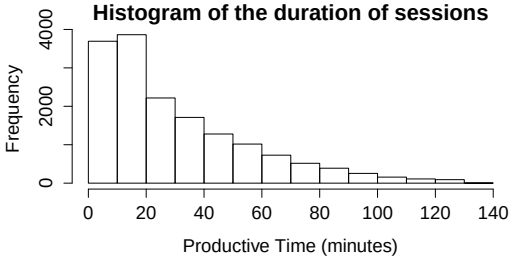


Figure 5. Statistics (left) and distribution of the productive time (right) of the sessions in UDC

4.5. Transformation

Once the events were classified and sorted, we split the dataset into sessions, defining a session as a period of registered work between interruptions of 8 hours or larger, as we did with Mylyn. We split the data by users to avoid a conflict between the last event of a user and the first one of another. After this, we obtained a total of 28,989 different sessions; but after a filter to keep only those with at least 30 minutes of productive time (as we did with the Mylyn dataset), we ended up with 15,825 useful sessions, in contrast with the 4,284 we had from Mylyn. On average, every session contains 154 edition and selection events, and this number is positively correlated (0.81) with the size of the session. This is an expected result, as having more time consequently increases the number of events in the session. Additionally, it is important to remark that the number of events in a session do not alter the edit or selection events per minute.

To identify an the interruptions on every session, we considered the same definition we used with Mylyn, where an interruption is a segment of 3 minutes or more with no activity registered.

At this point, we had the same format of sessions with both datasets. In the Figure 5 we can see statistics about the sessions (left) and a histogram (right) of the duration of the sessions in UDC, which ignores outliers by setting a threshold of three times the standard deviation.

4.6. Metrics Used in This Study

Before introducing the results about the first research question, it is convenient to present the metrics used in this study. We use the following six metrics to measure interruptions and productivity, while controlling for the unproductive time in a session, the length of time of the session itself, and the efficiency of the development that took place during the session.

Metrics characterizing interruptions:

- *Number of interruptions*: counts all the interruptions that occur in a development session.
- *Duration of interruption*: it is the time duration in minutes of the each interruption.

Metrics characterizing productivity and activity:

- *Productive work time:* the duration of a development session, subtracting the duration of all the interruptions present in the session, to control for inactivity.
- *Number of edits per minute:* the total number of edits events, divided by the productive work time to control for length of the session. This is an indicator of user activity during the session.
- *Number of selections per minute:* is the total number of selection events, divided by the productive work time. Also an indicator of activity.
- *Edit ratio:* the number of edits divided by the sum of edits and selections, as used by Kersten and Murphy [17]; an efficient developer spends less time exploring code and more time editing it.

We measure the effects on productivity using mainly the edit and selection events per minute as indicators. The edit ratio might be also a good indicator, under the assumption that a productive programmer will execute more editing activities than selection or navigation events. However, this

DOI: 10.1002/smr

Table VIII. Correlation analysis of the metrics used in the study.

metrics	correlation
Traces without interruptions	
Edit/min vs Sel/min	0.25
Edit/min vs Edit ratio	0.65
Sel/min vs Edit ratio	-0.38
Traces with interruptions	
Edit/min vs Sel/min	0.35
Edit/min vs Edit ratio	0.60
Sel/min vs Edit ratio	-0.32

is not always the case, given that depending on the role of the worker, selection events can be equivalent to the usage of a designing or reporting tool.

An important reason to consider the edit ratio is that having an interruption requires doing activities that might increase the execution of selection events, reducing the edit events. For example reading the code, navigating around files and starting a debugging sessions are activities heavily loaded of selections events and are common activities after an interruption, according to observational studies. The edit ratio can tell us when there is a reduction of edit events due to (mainly) selection events.

We did a correlation analysis involving the three metrics used (Table VIII, classifying the data between traces with or without interruptions. We find that the Edit/min and Sel/min metrics have low correlation coefficients, which is the expected result. The same happens between the Sel/min and Edit ratio metrics with the data of both kind of traces. The Edit/min and Edit ratio correlation coefficient is higher in both groups, but not high enough to not consider the latter, for it tells something different from the user activity.

5. RQ1: RELATIONSHIP BETWEEN INTERRUPTIONS AND PRODUCTIVITY

5.1. Relation between Interruptions, and Edits and Selections

As mentioned above, we use the metrics of edit, selection, and edit ratio as indicators of productivity. We first examine the number of edits and selections, and how their distribution varies in function of the number and type of interruptions.

We split the data in five groups: The first group contains all the sessions without interruptions (*none*). For the others groups, we have considered four ranges of number of interruptions delimited by their quartiles; the results differ between Mylyn and UDC, existing more interruptions per session in the former (Table IX). Then, for each group, we display the distribution of events per minute and edit ratio via boxplots. We can see the results with Mylyn in Figure 6), and for UDC in Figure 7.

In the results with Mylyn, we observe a large difference between the sessions without interruptions and the ones who do. Further, we observe that the rate of events per minute decreases slightly when the session has more interruptions. Therefore, we can intuit that the relationship between number of interruptions and the productivity indicators that are edits and selections, tends to be inversely proportional.

With UDC, the results show a similar pattern in comparison with the results with Mylyn. When facing zero interruptions the editions and selections per minute are greater than with one or more interruptions, and as more interruptions occur this metrics gradually decrease. The difference between sessions with no interruptions and at least one is less marked in the edit per minute metric, but the selections per minute metric is more alike to the Mylyn results.

Beyond visual inspection, we also quantify the statistical and the practical significance of these observations. First, in the results with Mylyn, all the differences observed are statistically significant

DOI: 10.1002/smr

Table IX. Thresholds used to group sessions based on their number of interruptions

	25%	50%	75%
Mylyn	3	5	10
UDC	2	4	7

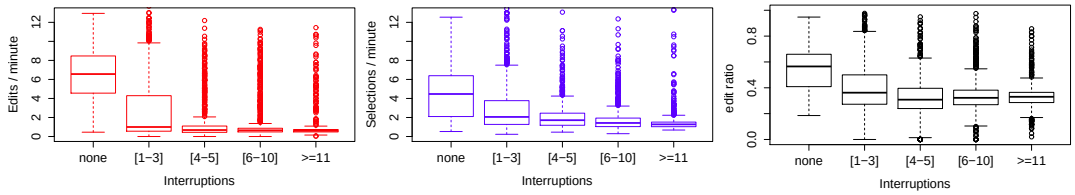


Figure 6. Boxplots showing the relation between the number of edits and selection events per minute, the edit ratio, and the number of interruptions in Mylyn.

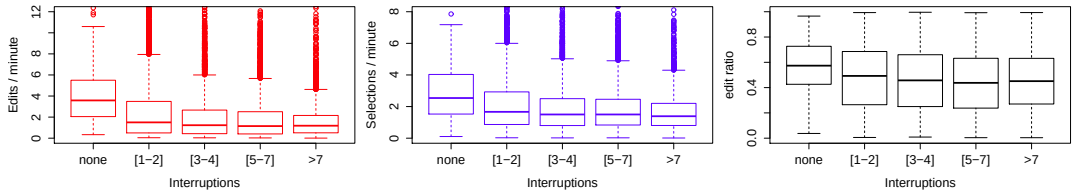


Figure 7. Boxplots showing the relation between the number of editions and selections per minute, the edit ratio, and the number of interruptions in UDC.

with very low p-values (see Table X) according to the Mann-Whitney U-test. This is not surprising, given the shape of the boxplots and the size of the samples.

More importantly, we used *Cohen's d* to measure the practical significance of these results in term of effect size [39]. *Cohen's thresholds* are defined as follows: trivial (< 0.2), small ($\langle 0.2 - 0.5]$), moderate ($\langle 0.5 - 0.8]$) and large effect (> 0.8). As shown in Table X, we note that the effect size of the interruptions over the number of edits by minute is (very) large. In selections, the effect is moderate for sessions having up three interruptions, and large to very large for sessions with over four interruptions. This reinforces our impressions that interruptions and user activity follow inverse relationships, and that they are quite pronounced.

On the other hand, with the results in UDC, an important difference is that the effect overall appears to be less pronounced in UDC and we can see it on the effect size tests shown in the Table XI. Most of the observed effects are medium, while in the Mylyn dataset the effects were large. Note that despite the differences in effect sizes, the differences are still significant and it is possible to reach the same conclusions.

5.2. *Effect on the Edit Ratio*

Finally, we want to know the effect of the interruptions over the ratio of edits in each session, as this is the often seen as a better indicator of productivity than raw activity, since the programmer spends less time navigating the source code in search of information, and more time actively editing it [17].

We first analyze the relationship between edit ratio and number of interruptions in Mylyn (Figure 6). We observe that the edit ratio decreases when the session has more interruptions: the effect is pronounced between session that do not have interruptions and ones that do, and is more subtle as the number of interruption grows. A look at the practical and statistical significance of these results (Table X, bottom) show that the results are (unsurprisingly) statistically significant, and that the observed effect sizes are large to very large.

DOI: 10.1002/smr

Table X. Effect size and significance of the relationship between number of edit per minute, selection per minute, edit ratio, and number of interruptions on Mylyn data

	none	≤ 3	$[4-5]$	$[6-10]$	≥ 11
Edits					
mean	6.29	2.59	1.55	1.29	0.91
U-test	$\hookrightarrow$	$<$ 2.2e-16			
Cohen's d	$\hookrightarrow$	**1.23**	**2.02**	**2.37**	**3.31**
Selections					
mean	4.73	2.93	2.26	1.79	1.54
U-test	$\hookrightarrow$	8.1e-13	$<$ 2.2e-16		
Cohen's d	$\hookrightarrow$	**0.66**	**1.17**	**1.93**	**1.78**
Edit ratio					
mean	0.55	0.39	0.33	0.34	0.33
U-test	$\hookrightarrow$	4.2e-14	$<$ 2.2e-16		
Cohen's d	$\hookrightarrow$	**0.84**	**1.33**	**1.48**	**2.12**

Table XI. Effect size and significance of the relationship between number of editions per minute, selections per minute, edit ratio, and the number of interruptions on UDC data

	none	≤ 2	$[3-4]$	$[5-7]$	≥ 7
Edits					
mean	3.97	2.18	1.82	1.70	1.59
U-test	$\hookrightarrow$	$<$ 2.2e-16			
Cohen's d	$\hookrightarrow$	**0.58**	**0.88**	**1.05**	**1.35**
Selections					
mean	3.10	2.93	1.80	1.77	1.63
U-test	$\hookrightarrow$	$<$ 2.2e-16			
Cohen's d	$\hookrightarrow$	**0.44**	**0.66**	**0.81**	**0.85**
Edit ratio					
mean	0.68	0.56	0.55	0.54	0.57
U-test	$\hookrightarrow$	$<$6.9e-06	$<$3.6e-09	$<$5.1e-11	$<$5.1e-09
Cohen's d	$\hookrightarrow$	**0.37**	**0.46**	**0.49**	**0.55**

As for UDC, the edit ratio behaves similarly in comparison with Mylyn, where the median changes accordingly with the number of interruptions, decreasing when there are more and reaching the maximum when there are none. In addition, the size and form of the boxes indicate more variation in this results and the differences between groups is more subtle, judging by the small changes in the mean values.

Adding this to our previous result, we observe that both the user activity (in terms of raw quantity of edits and selections per minute) and the user productivity (in terms of edit ratio), follow an inverse relationship with the number of interruptions. This finding agrees with the previous literature on the harmfulness of multitasking, work fragmentation, and interruptions. Furthermore, the effect sizes are between medium to very large.

6. RQ2: RELATIONSHIP BETWEEN DURATION OF INTERRUPTIONS AND PRODUCTIVITY

In this section we analyze whether or not the interruption duration is a factor in the relationship between interruptions and developer productivity. Under the hypothesis that prolonged interruptions require more time to recover, we have built three groups of sessions based on the proportion of long interruptions. As we described previously about the kind of interruptions we consider, a prolonged interruption are those whose duration is $\geq$12 minutes, and could be due to an extended interruption, a coffee break or meeting.

The proportion of long interruptions is calculated as the sum of interruptions longer than 11 minutes, divided by the total of interruptions in that session. Then, we obtain the quantiles of the distribution of this proportion and use them to create 3 groups described like the following:

DOI: 10.1002/smr

Table XII. Median for the edits per minute, selections per minute and edit ratio according to the number of interruptions and the proportion of prolonged interruptions in UDC.

	Number of interruptions			
Proportion of long int.	none	[1-3]	[4-6]	>6
Edits per minute				
low	3.58	2.52	1.45	1.36
medium	3.58	1.25	1.15	1.27
high	3.58	1.25	1	0.95
Selections per minute				
low	2.54	2.15	1.6	1.44
medium	2.54	1.5	1.5	1.41
high	2.54	1.55	1.45	1.36
Edit ratio				
low	0.57	0.54	0.47	0.46
medium	0.57	0.45	0.44	0.46
high	0.57	0.46	0.42	0.44

- *low*: the first group consists of sessions where the proportion of prolonged interruptions is <0.30, which includes sessions with proportion equal to zero, that is sessions with only short interruptions. These sessions constitute 0000% in Mylyn and 20.8% in UDC, from the total of sessions in both datasets.
- *medium*: the second group consists of sessions with a proportion between 0.30 and 0.50. These sessions constitute 0000% in Mylyn and 18.9% in UDC.
- *high*: the sessions in this group have a proportion >0.50, which are the 0000% in Mylyn and 60.1% in UDC.

Following the conclusions on RQ1, the number of interruptions may affect the results if we ignore that variable. So, we split every group in accordance to the number of interruptions; this time, instead of taking the quantiles of the distribution we took the terciles. Thus, for the three metrics, we created the groups low, medium and high, and every group was split in other three groups according to the terciles of the distribution of the number on interruptions.

The results with Mylyn show...

The Table XII shows the median of the metrics when grouping the sessions by the number of interruptions and the proportion of prolonged interruptions. In this context, it is better to have a low number of interruptions (group [1-3]) and a low proportion of prolonged interruptions, and the results agree with this statement. For example, when having between 1 to 3 interruptions and a low proportion the metrics show higher values when compared to having > 6 interruptions and a high proportion of prolonged interruptions. Moreover, if we focus on a category we can see a gradual decrease as the number of interruptions increases, which lead us to conclude that the proportion of prolonged interruptions has an effect on the developer productivity.

The difference between categories seems to decrease when reaching higher values. There is a clear difference between the low and medium categories of proportions, but the difference between medium and high is minor. The same happens when comparing the categories [1-3] and [4-6] of the number of interruptions, and the categories [4-6] and >6. In other words, as the value of the metrics get higher, the negative effect (according to these metrics) changes at a very low rhythm.

The statistical tests show that...

DOI: 10.1002/smr

7. RQ3: LOCAL RELATIONSHIPS BETWEEN INTERRUPTIONS AND PRODUCTIVITY

The impact of work fragmentation could be more noticeable in the immediate minutes before and after an interruption occurs. On one hand, after an interruption the programmer carry out activities meant to recover the lost mental state, like reading the code, debugging, reading notes and cues, and more resumption strategies [13]. Depending on the interruption, the recovery could last about 15 minutes, according to the reports from field studies [6, 44]. In the data from UDC, the sessions with duration of at least seven hours (commonly the working time of a software developer) have 11 interruptions in average; if we consider a recovery time of 15 minutes, the total time required to recover from the interruptions can be of about two hours, which is about one quarter of the whole session.

On the other hand, before the interruption there is a preparation phase when the interruption is imminent or expected, e.g. lunch time, a meeting and other scheduled events. In this phase the programmer might leave notes or comments in the code to recover the context after the interruption [14]. Also, when the interruption happened because of a problem found by the programmer usually he tries to solve it by reading the code, using the debugger or reading comments in the control version tool; after this resources are depleted, the next option is to ask to teammates or other external resources [40], which generates an interruption of work. The activities prior to the interruption are commonly associated with selection and navigation tasks, like reading the code and navigating to another classes or files to try to find an answer to the problem.

In this research question we expect to see the effects on productivity around an interruption due to the activities previously described.

7.1. Generic Sessions

In order to better understand the relationship between work fragmentation and productivity, we need to delve deeper and perform local analyses of the development sessions. In the first step, we wanted to summarize how the activity of users is distributed over time in sessions which have no interruptions, short interruptions, and prolonged interruptions. For this, we created the following groups:

- *none*: the first group contains sessions with no interruptions.
- *short*: the second group consists of sessions that only have short interruptions ($<$ 12 minutes of duration).
- *prolonged*: the third group consists of the remaining sessions, which have at least one prolonged interruption ($\geq$ 12 minutes of duration).

We performed this grouping in both Mylyn and UDC, obtaining the proportions shown in Table XIII. In both datasets the proportion of sessions with at least one long interruption is much larger than the rest, and there are a few sessions without interruptions.

Table XIII. Proportion of sessions with short interruptions, at least one prolonged interruptions and none

	none	short	prolonged
Mylyn	2%	18%	80%
UDC	1%	6%	93%

To summarize the development sessions, we have resized each time series to a single $size$, using local means. We have used $size = 10$, that is, dividing each session in 10 chunks of equal time, since because it yielded better global visualization (better uniformity) of the user interaction along a session. For each chunk, we choose the median value of edits and selections for each group, and compose one summary time series for each group; these are the time series displayed in Figure 8, for Mylyn, and Figure 9, for UDC.

We observe that the median activity in sessions with interruptions is less than in sessions without interruptions, mainly in the edit frequency. Moreover, the edit frequency exceeds the selection

DOI: 10.1002/smr

frequency in sessions without interruptions. The opposite occurs in sessions with at least one interruption—and is more pronounced when there is at least one prolonged interruption—, where the code navigation tends to exceed the frequency of edits. These results agree with our earlier results.

We also notice that the time series without interruptions have much more edits in the middle of the session than the others. This observation matches the hypothesis that developers need time to build their mental model for their task, and are less productive early in a session. This is similar to the edit lag in Parnin's study [13]: in sessions without interruptions, the edit lag is clearly visible at the start of the session. This behavior is not visible in sessions with interruptions; indeed, each session may have several edit lags (after each interruptions), and those would be equally distributed over the entire duration of the session, resulting in the "flat lines" that we see for these sessions.

These results support our previous observations at a finer level, and corroborate the literature saying that time is needed to start or get back on task. Also, we observe the same conclusions in the results with UDC, but there are some subtle differences. For instance, the distance between edit and selections in every chunk is shorter than the results with Mylyn, and during sessions with short interruptions the selections start being greater that the editions, but around the middle of the session the latest slightly recover, and finish below the selections again at the end of the session.

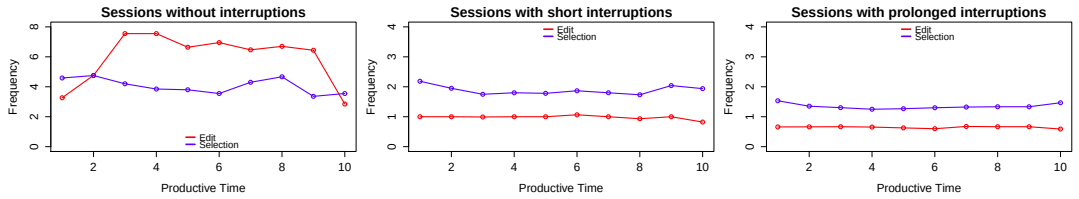


Figure 8. Global representation of a session in Mylyn, with: no interruptions (left); only short interruptions (middle); and at least one prolonged interruption (right).

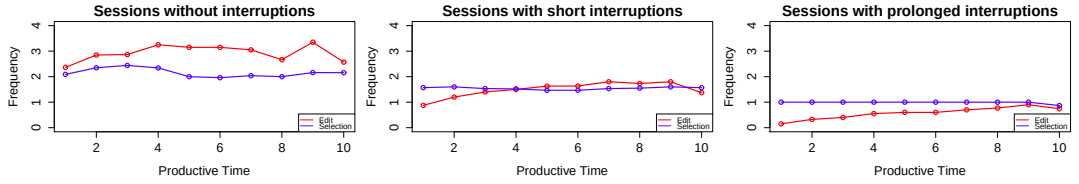


Figure 9. Global representation of a session in UDC, with: no interruptions (left); only short interruptions (middle); and at least one prolonged interruption (right).

7.2. *Local Analysis*

Having presented the global effect of the interruptions over the user productivity, we now focus on the local activity before and after interruptions. We take a maximum real time interval of 30 minutes around each detected interruption, obtaining a set of 26,988 time series subsequences in Mylyn and 97,984 in UDC. Then, we compute the median of all these subsequences as a generic local representation (Figure 10). We also plot with dashed lines the median values of edits and selections per minute in the sessions with interruptions in order to give more context to the observed values. Below we describe some observations:

- *In the center,* we find the interruption point. There is clearly a negative effect on the time series, as the area before and after the interruption is the area with the lowest activity. The activity is well below the median activity of time series with interruptions, showing that the effect is indeed more pronounced near interruptions. This happens in Mylyn and UDC.

DOI: 10.1002/smr

- *On the right,* the trend of the time series increases steadily. We hypothesize that the programmer is immersing again into the programming task, increasing progressively the activity as represented by the number of edits and selections. In Mylyn it reaches the median activity (in average) 12 to 18 minutes after the interruption, while in UDC it is around 6 to 12 minutes. It then rises further than the session median, which is not surprising, as we expect higher than median activity further away from interruptions.
- *On the left,* we observe that the number of events near the interruption also goes down well below the median value. This might be because of two reasons: the programmer could have found a problem while coding, who at first will try to solve it by reading the code, switching out from the IDE, navigating the call stack and debugging, before going to ask another teammate according to LaToza et al. [40]; this set of actions end up with an interruption and reduce the observed activity within the IDE. Also, when the interruption is imminent or expected, the programmer make use of different suspension strategies like writing physical notes, making a mental note or leaving a reminder cue on the code or window, as mentioned by Parnin and DeLine [14]. These activities reduce the activity before the gap and they are seen in the interaction data mostly as selection events. The latest can be better visualized in Figure 10 with the UDC data as a significant decrease of edit events, being below the selection events approximately 10 minutes before the interruption.

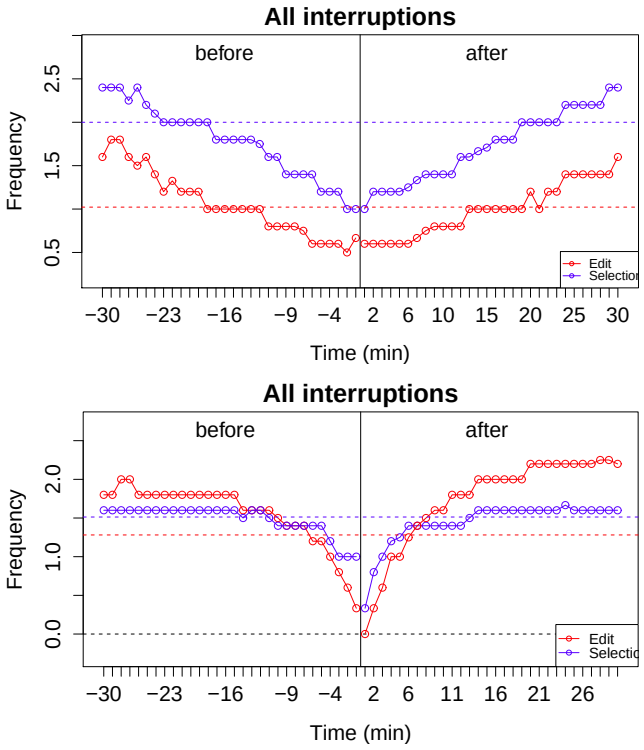


Figure 10. Local effect of an interruption in the user activity for Mylyn (top) and UDC (bottom)

We investigated whether the drop in activity local to interruptions was also accompanied by a drop in the edit ratio. We computed the edit ratio over slices of 5 minutes before and after each interruption (smaller intervals would be too sensitive to noise). The boxplots in Figure 11 show that the edit ratio drops slightly the closer we are to an interruption. The effect is small, but significant (see Table XIV). As a reference, the mean edit ratio for sessions with short interruptions is 0.4, so the edit ratio close to the interruptions are clearly lower in the 5 to 10 minutes around an interruption. This matches the observations Parnin made with the edit lag [13].

DOI: 10.1002/smr

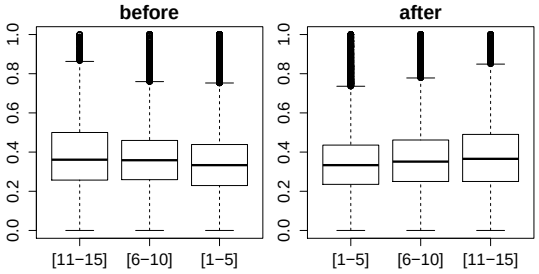


Figure 11. Ratio of edits around the interruption.

Table XIV. Effect size of the ratio of edits around the interruption.

	[1-5]	[6-10]	[11-15]
before			
mean	0.34	0.37	0.39
U-test	↪	$<$ 2.2e-16	
Cohen's d	↪	**0.16**	**0.24**
after			
mean	0.34	0.37	0.39
U-test	↪	5.67e-16	$<$ 2.2e-16
Cohen's d	↪	**0.14**	**0.22**

7.3. Patterns of Interruptions

Given the overall activity pattern we noticed in the local analysis, we hypothesize that there are several kinds of interruptions, matching the scenarios observed in the literature: actual interruptions distracting the programmer from the task at hand, and switching tasks in case of being stuck in the current task.

We hence looked for patterns in the interruptions. After applying clustering techniques over all the subsequences, we always found the formation of three recurrent patterns that show different effects of the interruption: neutral, positive and negative. The clustering was performed with the K-Mediods [41] technique, the Silhouette metric [42] to interpret and evaluate the results, the Dynamic Time Warping [43] as distance measure, and feature extraction techniques to reduce the dimensionality. For this reason, we classified empirically each interruption by its local effect.

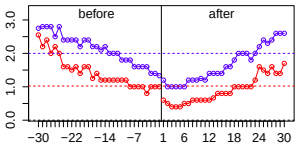
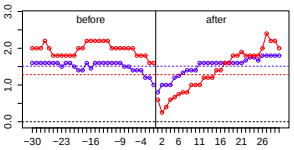
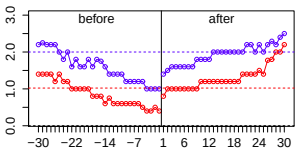
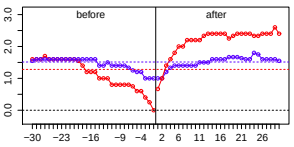
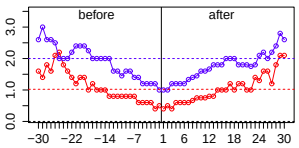
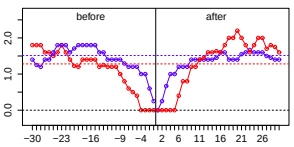
We did so by computing Cohen's d on the quantity of edits before and after the interruption. To obtain a significant effect, we need the presence of activity both before and after. Not all the interruptions meet this criteria however: some are located close to the start or the end of a session, or too close to another interruption. In total, 53% of the interruptions had 30 minutes before and after the interruption and were selected for the analysis. The remaining 47% of the interruptions can not be used on this analysis due to the lack of time around them. Table XV shows the applied thresholds and the results, accompanied with a typical example of an interruption in each category. This local analysis shows that there are indeed three well-defined groups of interruptions, with the two largest of them having clear effects on the activity in the session.

The results in UDC indicate that after a positive interruption, both metrics tend to increase at a high rhythm until reaching the global average value. The edits per minute recover quickly (approximately 5 minutes after the interruption), but the selections per minute take a little longer to reach the global average, and this metric tend to be below the average for an extended period. It is important to notice that before the interruption happens the editions per minute metric is below the average for approximately 16 minutes prior to the interruption.

In contrast, when facing the effects of a negative interruption the metrics take longer to reach the median. On the contrary to facing a positive interruption, the editions per minute metric is below the selections per minute, and the former takes longer than the latter to reach the average value. Also, the editions per minute metric is above the average for the 30 minutes before the interruption.

DOI: 10.1002/smr

Table XV. Local Effect of Interruption.

effect	pattern in Mylyn	pattern in UDC
negative: when the frequency of edit events decreases after the interruption (Cohen's $d < -0.2$)		
positive: when the frequency of edit events increases after the interruption (Cohen's $d > 0.2$)		
neutral: when there is no well defined effect before or after the interruption ($abs(d) <= 0.2$)		

In general, the effect of positive and negative interruptions is the inverse. The editions per minute describes better the effect and the selections do not show a major change. There is not a clear pattern during a neutral interruption.

Finally, we briefly report on the edit ratios for positive and negative interruptions (see Figure 12). We see distinct patterns as well: positive interruptions have a higher edit ratio after the interruption (in accordance with the hypothesis of a more efficient activity after having looked for missing information), while negative interruptions have a lower edit ratio after the interruption (in accordance with the hypothesis that the programmer may be rebuilding his context after an unwanted task switch).

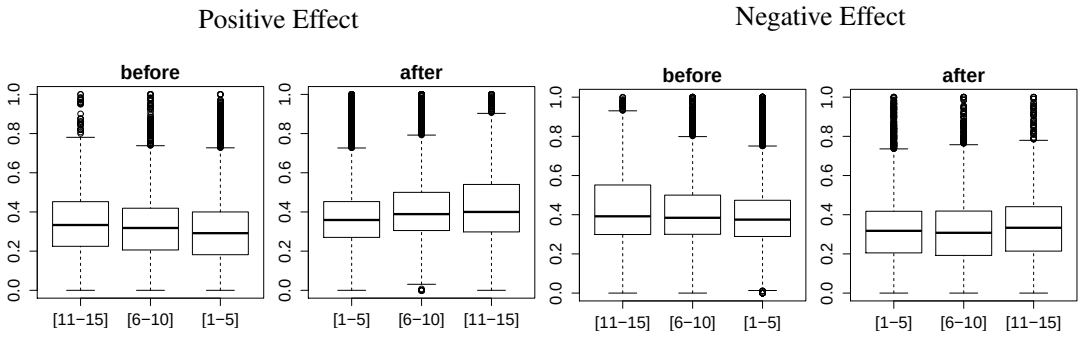


Figure 12. Ratio of edits for positive and negative local effect.

8. RQ4: RECOVERY TIME AFTER AN INTERRUPTION

8.1. Edition and selections during recovery time

The amount of information we have in Usage Data Collector allows us to analyze more specific sections on a programming session. We further investigated the immediate activities after an interruption that we name the *recovery time*, which is a time period taken by the programmer to

Prepared using smrauth.cls
DOI: 10.1002/smr

resume the interrupted work. We could not do this with Mylyn data, since the process involves a lot of filtering of interruptions, which left us with too few data points to analyze.

We established the recovery time as the 15 minutes period after an interruption, without considering the time consumed by the interruption itself. The length of the period was chosen according to the observations by Solingen et al. [44], who explain that although it is a variable difficult to measure, estimating 15 minutes for this phase is believed to be valid. They also consider the recovery time as the third (and last) phase of an interruption; the first one is the occurrence phase, or the start interruption, followed by the handling phase that contains all the activities required by the interruption.

Furthermore, Iqbal and Horvitz [6] conclude, after observations from field study, that the recovery time can last between 10 to 16 minutes, depending on the interruption. They speculate that this time is necessary to refocusing cognitive resources lost during the diversion.

In order to execute this analysis, it was necessary to make a selection of the interruptions that have a period after it of at least 15 minutes of recorded work, before another interruption occurs. After this query we obtained 5,684 valid interruptions from a total of 110,077.

From those valid interruptions, we split the time after the interruption into two phases:

- *Recovery time*: the period after the interruption of 15 minutes in length.
- *After recovery time*: the period after the recovery time until the next interruption.

Following this procedure, the recovery time was split in three phases of five minutes each to be able to analyze the evolution through this phase on detail. Also, the period after the recovery time was split into the following phases:

- *Before interruption*: a period of 10 minutes before the next interruption.
- *Peak*: the time between the recovery time and the before interruption phase.

Splitting the recovery phase into three phases allows us to analyze in detail the evolution of productivity and identify the immediate activities after the interruption, which will be helpful later. On the other hand, the before interruption phase represents the time where the productivity drops just before the next interruption, as seen in the results on RQ3. The peak phase is where the productivity should be reaching its maximum, as the programmers are not taking actions to retake the lost mental model or preparing for an interruption. The Figure 13 shows how the time after the interruption was split into phases.

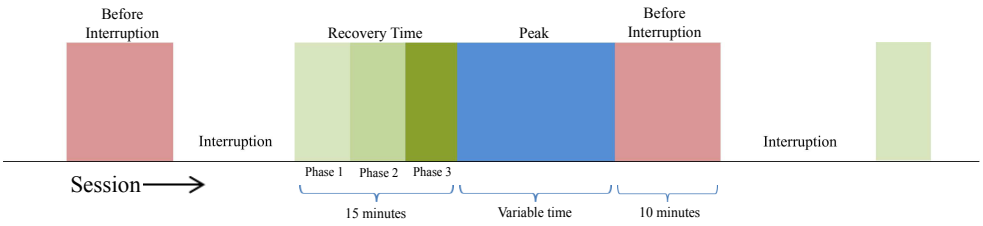


Figure 13. Graphic showing how the time after an interruption was split.

The duration of the peak varies among the phases between interruptions. On average, the peak lasts for 12 minutes, while the maximum registered is of 145 minutes. The distribution of duration of the peaks is reported in Table XVI.

Table XVI. Quartiles of the distribution of the duration of the peak phase.

quartile	25%	50%	75%
value (min)	3.38	8.05	16.31

We can see in the Figure 14 the evolution of the metrics through the phases. The three metrics show a tendency to increase and reach a maximum between the last phase of the recovery time and

DOI: 10.1002/smr

the peak of the session. Moreover, the fall of the productivity on the last minutes before the next interruptions agrees with the observations on the local analysis of the time around an interruption. From this we can conclude that the first five minutes after an interruption and the moment before the next one are the are the phases where productivity is more affected, while the phases around the peak are where the productivity is more favorable.

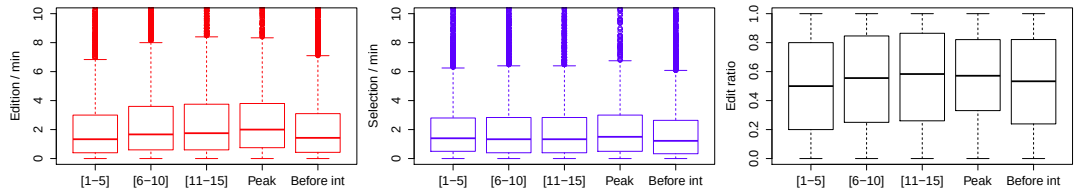


Figure 14. Boxplots showing the detailed evolution of editions, selections and edit ratio during recovery time and after recovery time.

9. RQ5: COMMON EVENTS DURING RECOVERY TIME

In RQ3, we saw different patterns of activity around positive and negative interruptions. In particular, the edits per minute were higher after a positive interruption; on the contrary they were lower after a negative interruption. The effect is even more marked in the immediate minutes after the interruption, as can be seen in the Figure 15.

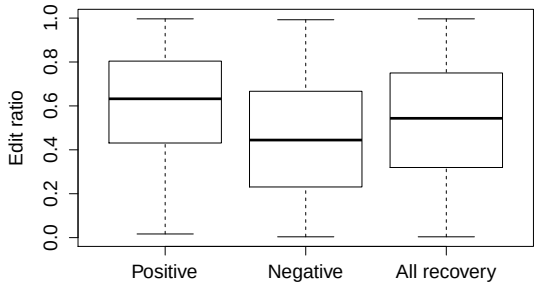


Figure 15. Boxplot of the edit ratio during recovery time after a positive or negative interruption, compared to the edit ratio overall.

The presence of this pattern and the important difference observed in Figure 15 led us to investigate the effect more closely. To this aim, we performed an exploratory study of the activities performed by programmers during the recovery time of an interruption. This is possible due to the high variety of events present in the UDC dataset, which can tell us a lot more about the type of activities developers do. Indeed, the UDC records individual commands that the developers issued in the IDE. The commands are very descriptive in the kind of activity that they represent.

The goal of the exploratory study then was to better understand whether the type of activities taken after an interruption were indicative of different types of activities for each type of session, and also whether these different kinds of activities supported our earlier hypotheses explaining why the two patterns of interruptions were present. Assuming that different commands represented different types of activity, the assumption was that the frequency of a given activity—as represented by the commands used to carry it—would vary between the different types of sessions.

A first challenge that we encountered was that, similarly to the earlier research questions, the high number of different commands made it hard to infer broader trends. We hence created an additional classification of the commands we encountered in the activity logs. In a similar manner to the first classification, we followed an iterative process involving two of the authors to classify the events

DOI: 10.1002/smr

in these categories. Differently to the first classification, we did not limit ourselves to a restricted number of categories: rather, we let the classification emerge from the iterative process. We ended up with 11 categories, as described in Table XVII.

To better understand the reasons of the difference between positive and negative interruptions, we did an analysis of the common events during recovery time, depending on the nature of the interruption. For this we created a new classification of events on a lower level that takes into consideration what kind of activity is performed by the programmer. The new classification contains the categories described in the Table XVII. This table also contains the frequency per minute of the more frequent event on that category.

A more detailed description of each category follows.

Category *high-nav* regroup events indicative of program exploration activities, such as switching between two files (usually classes), browsing a list of methods, using the outline tabs in Eclipse, etc. This category regroups most but not all usages of the views surrounding the main editor view, as well as menu items such as going to the declaration of an entity, browsing references to an entity, etc.

Category *file* regroup events related to file management, such as opening and saving files, refreshing its contents and editing its properties. This kind of activity is indicative of program editing in the sense of (mostly) saving files.

Category *edit-text* regroup most commands related to text edition, such as inserting or deleting characters, words, copy and paste, etc. This category is clearly indicative of program editing activities, as it regroups events that relate to the modification of low-level source code.

Category *debug* is concerned with events that occur during debugging sessions, such as setting and removing breakpoints, stepping in the source code, going backwards/forwards, stepping into the execution of a method, etc. This category is clearly related to program understanding.

Category *text-nav* regroups events related to text-level navigation, such as moving to the next word, to the beginning/end of lines, or up/down a line. If this category includes navigation activity, it is still related to low-level artifacts as it represents navigation over single files, and usually small distances (characters or lines). On the contrary, category *high-nav*, is related to the navigation between more abstract program elements (classes, methods, packages, interfaces), rather than low-level source code. As such, *text-nav* is more closely related to code editing, while *high-nav* is more closely related to program understanding.

Category *search* and *find-replace* are both concerned with search activities, although there are subtle differences. *search* regroups most kinds of search tools, such as text search, or searching code entities (types, packages, methods). As such, it is more to be seen as a high-level program navigation tool (the search results may span several files and are presented in a view that allows quick access to the entities—using such a view is classified in the *high-nav* category). On the other hand, *find-replace* concerns commands that are usually employed for text navigation and more particularly to perform editions (systematically replacing occurrences of a string with another). If the first category is related to program exploration, the second is more related to program edition. There is a caveat here, as it is possible to use *find-replace* commands only to find elements, which is more indicative of program exploration; however, the UDC dataset does not allow us to distinguish these usages.

Category *perspective* deals with events related to perspective change in Eclipse, such as switching from the coding to the debugging perspective, or vice-versa. Eclipse allows the user to program in different environments, like enterprise, mobile and Web. It also supports activities regarding other kind of roles, like testing, design and database management. Each of this environments and roles might require to open a different perspective, so having events regarding a perspective change is an indicative of the different roles that a user has. As such, we have no expectation as to whether this kind of activity is more indicative of program understanding or program editing.

Category *control* contains events related to version control (commit code, update code, comparing differences, resolving conflicts). This is hence related to finishing tasks, rather than editing or understanding code.

Category *refactoring* regroups usage of refactoring tools (rename, move methods, extract methods, etc). This is indicative of edition activities in the current piece of code, but it can also

DOI: 10.1002/smr

affect outer elements and require to verify the changes, implying low and high level navigation. Since automatic refactorings in the IDE are behavior-preserving, the need to verify the changes is lessened. As such, we consider these events to be more indicative of program editing than program comprehension.

Category *clean-build* regroup events related to build system usage (build the system, clean the build, run the build, the tests, etc). This is indicative of activities in which the programmer expects the program to be in a testable state, and not directly to program editing or understanding. This kind of events may also be automatically executed as part of a greater process. For example, when beginning a debugging session, the IDE might execute a clean and build action at first.

With these categories described, we continue with describing the hypotheses we made earlier when we observed the positive and negative patterns, namely that:

1. *Negative* interruptions—having higher overall activity before the interruptions rather than after it, and a lower edit ratio as well—could be indicative of actual interruptions. That is, a programmer is interrupted in his normal course of work, and needs to rebuild it's mental context after the interruption, performing additional program comprehension before resuming programming.
2. On the other hand, *Positive* interruptions—with higher activity after the interruption, and a higher edit ratio—could be indicative of information-seeking behavior. Programmers, noticing that they are encountering a problem that they are not able to fix on their own, may look for outside help (asking an expert, querying a question and answer site such as Stack Overflow, etc), and once a solution is found, resume their work in the IDE in a much more productive manner.
3. **Romain**What about debugging sessions?

We would then expect that certain types of interruptions would feature some events more frequently than other ones. For instance, after a negative interruption, we would expect more program navigation events (indicative of program comprehension), while after a positive interruption, we would expect more programming activity since the problem that the programmer encountered was solved. In short, we expect that for some categories of events, there will be differences between the types of events. We put this hypothesis to the test by calculating the frequency of categories of events after positive and negative interruptions, and also compared it with the overall frequencies of the events. We describe how we computed these frequencies in the following.

Table XVII. New category of events.

category	description	frequency
high-nav	High-level navigation through classes, windows and artifacts	9.1
file	Management of files (saving, closing, opening, etc.)	8.9
edit-text	Text edition events	8.1
debug	Events executed during debugging sessions	3.3
text-nav	Low-level text navigation	3.1
search	Text or code search	0.6
find-replace	Search and replacement of text or code entities	0.6
perspective	Change of perspectives in Eclipse	0.5
control	Source control events	0.4
refactoring	Refactoring events	0.3
clean-build	Events related to the build system (clean, build or run a project)	0.2

The first thing we noticed is that while some events are used by all users, some events are used heavily by a minority of users and not by others (this matches the observation of Murphy-Hill et al., and by ourselves, that many development tools are under-used [28]). For example, the event

DOI: 10.1002/smr

copyLineDown is more frequent than the *rename* event, but the former is only used by 2.4% of the users and the latest by 25.2% of them. As such, we calculated the frequency of event taking into account how commonly used they were.

We calculated the weighted average of every event based on the frequency of execution during the first phase of recovery time after a positive or negative interruption, and the percentage of users that make use of that event (so that events that are used by very few users do not distort the results). The weighted average was calculated as:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

where x_i is de frequency of execution of the event i and w_i is the percentage of users that uses the event. This balances the average of execution for events used by a small number of users.

To get a baseline, the same frequency calculation was made for the entire dataset, without distinction of the location of events in the session. We then compared the weighted average changes to have evidence of what kind of events are more often executed after a positive or a negative interruption. The Table XVIII shows the results, and is explained below.

Table XVIII. Weighted frequency of execution after positive or negative interruptions.

category	positive	negative	all
Most common events after a positive interruption			
edit-text	**34.75**	22.35	26.62
file	**10.56**	9.53	10.01
refactoring	**1.29**	0.84	1.01
perspective	**0.38**	0.30	0.35
Most common events after a negative interruption			
high-nav	7.84	**10.14**	8.74
find-replace	1.26	**1.47**	1.11
debug	3.08	**7.81**	6.35
search	0.57	**0.96**	0.73
clean-build	0.42	**0.62**	0.56
Less common events after a positive or negative interruption			
text-nav	14.20	12.89	**16.13**
control	1.21	1.75	**1.82**

After a positive interruptions the events belonging to the *edit-text* category, the *file* category, the *refactoring* and the *perspective* category, are more frequent than in the average. In addition, these categories of events are less frequent than the average after a negative interruptions. Of these four, the two categories with the largest differences between positive and negative interruptions are *edit-text* and *refactoring*. Both of these categories are much more clearly associated with edition activities than program comprehesion, and thus fit the previous hypothesis. Of the two other categories, *file* management is also related to program edition activities, why the role of the *perspective* category is still to clarify.

After a negative interruption, the *debug*, *search*, *high-nav*, *find-replace*, and *clean-build* are all more common than average. Of note, the *find-replace* category is also more common in positive interruptions. The three categories of *debug*, *search*, and *high-nav* are clearly associated with program comprehension rather than program edition. The *clean-build* category is not strongly associated with program comprehension, but it is however associated with debugging activities. The *find-replace* category which is more common in both positive and negative interruptions can be, as explained above, both used for find-and-replace text editions, and also for conventional search, which could explain its presence in both kinds of interruptions.

Finally, the *text-nav* and *control* categories are less common in both positive and negative interruptions than in the average. We were somewhat surprised that *text-nav* was less common in

DOI: 10.1002/smr

positive interruptions than the average, as we expected it to be associated with program edition behavior. We note that it is however more frequent in positive interruptions rather than negative interuptions. Additional exploration is necessary to understand the behavior of the *control* category.

Summing up, we can conclude that edition-related activities are indeed more frequent after a positive interruption, while comprehension-related activities are more frequent after a negative interruption. These categories of events agree with our previous hypothesis that positive interruptions are related to information seeking behavior: programmers would interrupt their IDE activities when encountering a roadblock, find the information elsewhere (from the internet, an expert, etc), and return to the IDE with an increased productivity. This also agrees with the literature [13, 40]. The categories common after nagative interruptions also agree with our hypothesis that negative interruptions are related to actual task switches where context needs to be rebuilt afterwards, an activity that involves program comprehension [11, 10].

After a positive interruption, we can see that the more common activities are text edition, refactoring, file management and, to a lesser extent, navigation around perspectives. On the other hand the navigation around classes, find and replace, searches, debug and clean and build are actions more common after a negative one. The navigation in the code and control version activities are more common after the recovery time, having a lower frequency during a positive o negative interruption. Of course, this exploratory study does not allow us to confirm these hypotheses. However, the evidence we have discovered so far does not disprove them, which means that our confidence in these hypotheses is higher as a consequence.

10. DISCUSSION

Comparison of the datasets Although they represent the same kind of information both Mylyn and UDC datasets have some peculiarities that make them unique that we discuss here.

The interaction data recorded by Mylyn is the basis for a recommendation tool, and is focused on Java projects. On the other hand, the interaction data recorded by the UDC is only used for diagnostic purposes, and is much more generic. For instance, the UDC has interaction data from Java, Web, and mobile developers, and more specific roles like back-end developers, testers and C++ or PHP users.

As the UDC was installed by default on all the Eclipse distributions from 2008 to 2010, the size of the information collected is several orders of magnitude larger than the Mylyn dataset, which was collected since then on a few dozen of developers who installed it voluntarily and contributed to the Mylyn and Eclipse projects. This makes of the UDC data a better source of information to analyze phenomenons on a wide scale.

The UDC and Mylyn also offer different levels of details: while the UDC records fine-grained type of events, Mylyn records the entity that were affected by the events (Java packages, classes, and methods). On the other hand, Mylyn has essentially two types of events of interest (edits and selections), while the UDC has no information about the entities that were affected.

Comparison of the results We conducted a replication of the results of RQ1, RQ2 and RQ3 with the additional dataset. If we noted similarities in the observations across both datasets, it is important to discuss the differences in the size of the effects that were observed: the overall effects observed were larger for the Mylyn dataset than for the UDC dataset.

The fact that all the observations go in the same direction lends us more confidence in our conclusions. However, the reduced magnitude of the effect gave us pause. We can think of several reasons for this behavior:

- The models are not directly comparable: Mylyn records edition and selection events directly, while the UDC records a broader array of user activity. As such, our classification of UDC events as edits, selections, or other, does not exactly replicate the way the Mylyn model operates. It is possible that some events are more frequent in the UDC than in Mylyn, for instance. We debated several ways to fine-tune our model of the UDC data, including

DOI: 10.1002/smr

weighting some UDC events more than others, but ultimately decided against doing so. As the models are not directly comparable, we primarily focus on intra-model comparisons.
- The (comparatively) small size of the Mylyn dataset may have favored outliers: since there is data from dozens of developers only in the Mylyn dataset, a few outliers may have affected the results. The UDC model is however based on 1,000 UDC users.
- Non-java users in the UDC: A small portion of UDC users were not primarily programming in Java; some were using C++, PHP, web frameworks, or a report generation plugin known as BIRT. These may have had an impact on the results.

The effects of multitasking Certainly, work fragmentation is not a synonym of lower productivity rates. Multitasking can allow the worker to use dead periods of the project where there is a decrease of the required activities and do tasks related to other projects, increasing the efficiency and productivity [8]. At some level, switching between tasks can provide the needed arousal and alertness to improve the productivity, but higher levels of multitasking can be detrimental and cause a negative effect. Moreover, as the number of switches between tasks increases, the effectiveness degrades in a linear fashion [9].

Furthermore, the results indicate a clear diminish of productivity when facing interruptions, but with a high number of interruptions the change of productivity stops decreasing. This could be reflecting what observational studies tell us about multitasking, which is that with higher number of interruptions (increased multitasking) the productivity stops being affected and might increase. The latter actually occurs in the results but the effect is too subtle. We intend to reflect the effects of work fragmentation by classifying the interruptions into negative or positive, and in the results we can see a difference in productivity and the kind of activities performed.

Comparison with observational studies Our theoretical foundations are based on observational studies performed on information workers' development environments [5, 6, 40, 14]. These findings helped us understand the response to work fragmentation according to the recorded user activity, and it was possible to made a relation between the user's data and the phenomena that occur on the working places described in the studies.

Observing the user activity throughout a working session might be the best way to understand the effect of events that occur on the working place, like work fragmentation. However, implementing a study of this kind can be very expensive, for the time and material need. For example, to have the same amount of information we have with UDC it would require to monitor the user activity for a year or two and have a record of all the activities performed, like the number and duration of the interruptions. Moreover, this kind of research require on-site studies, exhaustive monitoring and interviews, which in some cases could be expensive or difficult to perform.

On the other hand, analyzing interaction data is a convenient way to understand almost the same kind of events on the environment. Usage data describes the history of activities of the programmer and it is possible to analyze the effect of events, as shown in this work. The problem with this is the lack of context of the events and, in our case, the lack of information of the programmers to enrich the results. Hence, we can not guarantee that an interruption was an actual interruption of work; due to the definition of an interruption, it could also be a change to a Web design tool, server configuration tool or the browser, which are common activities in Web developers and not necessarily detrimental to productivity.

The best might be to find a middle ground between observational activities and analysis of usage data. It is helpful to enhance the results using information provided directly by the participants via interviews, like the work done by Fritz et al. [18].

DOI: 10.1002/smr

11. THREATS TO VALIDITY

The main threats to the validity of our study are construct validity threats due to the operational nature of the data recorded by Mylyn and the UDC [1]. Most of them are mentioned above but recalled here for convenience.

For the Mylyn dataset, the main threat to validity is due to the exclusion of development session with aggregated information. We deemed that disaggregating the data as was done in other work [21] was not appropriate as we do not know the exact distribution of aggregated events in time, which is very important for this study. We thus elected to filter out part of the data. We have not found evidence of bias due to this but it may still be present.

Another threat comes from the metrics we use. Our choices are limited by the data recorded by Mylyn. We believe our three indicator (edits per minute, selections per minute, and edit ratio) provide a reasonable measure of productivity (especially with the addition of edit ratio, used in other work [17]). However we can not exclude that other indicators of productivity (such as actual changes [45]) may yield different results.

As we mentioned before, the edit ratio has been used in previous work as an indicator of productivity, but it doesn't describe well the true level of productivity of a programmer. The edit ratio is the amount of coding activities that the programmer performed in comparison with the other activities, but a productive programmer may not solely code; he could have other important tasks like managing a database, Web design and testing, which are activities that are not reflected as edit events. This is a big problem in the UDC data, for this information was captured from people that could be Java developers, Web developers, mobile developers, students, etc., and from this roles the only one that could use well the edit ratio as an indicator of productivity is the Java developer, if his only task is to work in the IDE. The others are prone to spend time (maybe most of the time) using graphical tools like Web designers and simulators, and this events are mainly represented as selection events.

We defined several thresholds empirically: 8 hours of inactivity to split a session in sub-sessions, 30 minutes as the minimum duration for a session, 3 minutes for the minimum duration of a interruption and $\geq$ 12 minutes for prolonged interruptions. We also tested with other close values and obtained similar results. However, a large variation of these thresholds might significantly alter the results.

We acknowledge that our separation of sessions in short and long interruptions is not perfect, as it is based on the presence of at least one long interruption, and nothing more. Other factors present in the group of sessions with at least one long interruption may contribute to the observed effect (for instance, these sessions may also have more interruptions overall).

The study has threats to its statistical conclusion. In particular, correlation does not necessarily implies causation: the drop in productivity may not be due to work fragmentation but to other factors (such as task difficulty). The fact that our results agree with the existing literature does help in this respect. We note that our local analysis uncovered two well-defined patterns around gaps of inactivity: ones that we hypothesize are actual interruptions (negative effect post-interruptions), and the ones that we hypothesize are more likely information-seeking activities (positive effect post-interruption). This result points to two different effects and is in need of further study; the additional evidence uncovered in the UDC does support this hypothesis, since different activities appear to take place in both types of interruptions. As stated in the introduction, we were careful not to conclude that every gap of activity is an actual interruption.

We used Cohen's d, a parametric effect size measure, as it has defined thresholds allowing easier interpretation of the results. Other effect size measures such as the common language effect size [46] or A_{12} [47] may slightly alter the discussion.

This study also has threats to external validity. The gathered data corresponds to a limited group of programmers, which use both Mylyn and Eclipse regularly. The set of evolution tasks considered is a subset of the ones present on the Eclipse bug report website. It may be biased one way or another. One source of bias is the impact of Mylyn itself: Mylyn has been shown to facilitate task switching and to increase the edit ratio of its users compared to non-users. In the context of this

DOI: 10.1002/smr

study, we hypothesize that Mylyn could reduce the effect of work fragmentation: in other words, the observed effects may be larger for non-users. Additional studies may alleviate these threats and confirm or infirm the previous hypothesis. Our additional dataset (the UDC) comes from a much larger and more varied sample of users, and does show similar behavior, which raises confidence in our results.

The same threats about the metrics, implications and statistical conclusions apply on the results observed with the UDC. Additionally, our classification of events is a possible threat, as it was based on our own criteria of what should be an Edition and Selection event. We took this liberty due to the lack of an actual pre-classification, and on the other hand the extra information available on UDC events compared to Mylyn. We took special care when establishing the classification, using an iterative process involving two authors of this paper. We also ran tests of what kind of data was recorded for specific usage scenarios of the IDE in order to better understand the nature of the dataset when particular events were unclear.

The workbench activation and deactivation events were a particular issue, for they can be executed under many circumstances. We performed live test to understand how these events work and gain confidence about their precise behavior. For instance, we found that they were reliable markers of when the user leaves and returns to Eclipse—and we used this fact to accurately detect interruptions—but we had more uncertainty on other scenarios. Therefore, we decided not count them as "system" events, and thus only use them to define an interruption or the end of a session.

12. CONCLUSIONS AND FUTURE WORK

This paper presented an empirical study on the prevalence of work fragmentation in software evolution tasks, and its relationship to developer productivity. The study was based on the Mylyn dataset of software evolution tasks, where work fragmentation is indicated by gaps of activity (interruptions) in the IDE, and productivity is measured in terms of the number of edits, selections, and the edit ratio. It was then complemented with data coming from the Eclipse Usage Data Collector (UDC) dataset, which offered us to complement our previous findings by analysing a larger sample of users, with a finer-grained classification of events.

We analyzed several thousands software evolution tasks, originating from several dozens of developers. Our global analysis found an inverse relationship between number and duration of interruptions and all three of our productivity indicators. These findings agree with the literature on information workers and software developers.

A subsequent local analysis around interruptions confirmed these results as the activity around interruptions was found to be lower than average. This analysis also found two well-defined patterns around interruptions: interruptions yielding a negative effects (consistent with a real-life interruption involving an expensive context switch), and interruptions with positive effects (consistent with information-seeking behavior). Further studies are necessary to expand on these findings.

With the Usage Data Collector dataset, we replicated our previous study to compare the results obtained with Mylyn. We found the same observations on the relationship of interruptions and productivity; even though the observed effects are less marked, we reach similar conclusions.

The UDC data also allowed us to extend our study with two additional research questions related to the recovery time after an interruption, and the two patterns of interruptions we observed.

We first analyzed the recovery time in detail, finding a higher productivity when the interruption is positive and we identified a positive evolution of the metrics as the programmer advances through this stage, reaching a peak of productivity between the last minutes of the recovery time and 10 minutes before the next interruption happens.

In addition, we found that after a positive interruption the most common events during the first minutes of the recovery time are those with an assertive intention such as edition, refactoring and saving files. In contrast, after a negative interruption the most common events are debugging, searching and navigation around classes, which tell us about a doubtful and uncertain behavior from the programmer. These findings support out hypotheses that positive interruptions are due

DOI: 10.1002/smr

to information-seeking, while negative interruptions are due to context rebuilding after an actual interruption.

REFERENCES

1. A. Mockus, "Is mining software repositories data science? (keynote)," in *Proceedings of MSR 2014*, 2014, p. 1.
2. L. Zou and M. W. Godfrey, "An industrial case study of program artifacts viewed during maintenance tasks," in *Proceedings of WCRE 2006*, 2006, pp. 71–82.
3. V. M. González and G. Mark, ""constant, constant, multi-tasking craziness": managing multiple working spheres," in *Proceedings of CHI 2004*, 2004, pp. 113–120.
4. G. Mark, D. Gudith, and U. Klocke, "The cost of interrupted work: more speed and stress," in *Proceedings of CHI 2008*, 2008, pp. 107–110.
5. G. Mark, V. M. González, and J. Harris, "No task left behind?: examining the nature of fragmented work," in *Proceedings of CHI 2005*, 2005, pp. 321–330.
6. S. T. Iqbal and E. Horvitz, "Disruption and recovery of computing tasks: field study, analysis, and directions," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2007, pp. 677–686.
7. M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *Proceedings of CHI 2004*, 2004, pp. 175–182.
8. S. Aral, E. Brynjolfsson, and M. Van Alstyne, "Information, technology, and information worker productivity," *Information Systems Research*, vol. 23, no. 3-part-2, pp. 849–867, 2012.
9. R. F. Adler and R. Benbunan-Fich, "Juggling on a high wire: Multitasking effects on performance," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 156–168, 2012.
10.
11. R. Minelli, A. Mocci, M. Lanza, and T. Kobayashi, "Quantifying program comprehension with interaction data," in *Quality Software (QSIC), 2014 14th International Conference on*. IEEE, 2014, pp. 276–285.
12. A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of ICSE 2007*, 2007, pp. 344–353.
13. C. Parnin and S. Rugaber, "Resumption strategies for interrupted programming tasks," *Software Quality Journal*, vol. 19, no. 1, pp. 5–34, 2011.
14. C. Parnin and R. DeLine, "Evaluating cues for resuming interrupted programming tasks," in *Proceedings of CHI 2010*, 2010, pp. 93–102.
15. W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the comprehension of program comprehension," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 4, p. 31, 2014.
16. A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of FSE 2014*, 2014.
17. M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of FSE 2006*, 2006, pp. 1–11.
18. T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 341–350.
19. T. Fritz, J. Ou, G. C. Murphy, and E. R. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of ICSE 2010*, 2010, pp. 385–394.
20. G. C. Murphy, M. Kersten, and L. Findlater, "How Are Java Software Developers Using the Eclipse IDE?" *IEEE Software*, vol. 23, pp. 76–83, 2006.
21. A. T. T. Ying and M. P. Robillard, "The Influence of the Task on Programmer Behaviour," in *Proceedings of ICPC 2011*, 2011, pp. 31–40.
22. T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *Proceedings of ESEC/FSE 2011*. ACM, 2011, pp. 311–321.
23. R. Robbes and D. Röthlisberger, "Using developer interaction data to compare expertise metrics," in *Proceedings of MSR 2013*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 297–300.
24. Z. Soh, F. Khomh, Y. Guéhéneuc, G. Antoniol, and B. Adams, "On the effect of program exploration on maintenance tasks," in *Proceedings of WCRE 2013*, 2013, pp. 391–400.
25. Z. Soh, F. Khomh, Y. Guéhéneuc, and G. Antoniol, "Towards understanding how developers spend their effort during maintenance activities," in *Proceedings of WCRE 2013*, 2013, pp. 152–161.
26. F. Bantelay, M. B. Zanjani, and H. H. Kagdi, "Comparing and combining evolutionary couplings from interactions and commits," in *Proceedings of WCRE 2013*, 2013, pp. 311–320.
27. M. B. Zanjani, G. Swartzendruber, and H. H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," in *Proceedings of MSR 2014*, 2014, pp. 162–171.
28. E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *Software Engineering, IEEE Transactions on*, vol. 38, no. 1, pp. 5–18, 2012.
29. H. Liu, Y. Gao, and Z. Niu, "An initial study on refactoring tactics," in *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*. IEEE, 2012, pp. 213–218.
30. C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 36.
31. E. Murphy-Hill, R. Jiresal, and G. C. Murphy, "Improving software developers' fluency by recommending development environment commands," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 42.
32. Y. Yoon and B. A. Myers, "Capturing and analyzing low-level events from the code editor," in *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. ACM, 2011, pp.

DOI: 10.1002/smr

25–30.
33. G. Khodabandelou, "Mining intentional process models," Ph.D. dissertation, University of Paris-Est, 2014.
34. W. Maalej, T. Fritz, and R. Robbes, "Collecting and processing interaction data for recommendation systems," in *Recommendation Systems in Software Engineering*, 2014, pp. 173–197.
35. U. M. Fayyad, G. Piatetsky-shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *Ai Magazine*, vol. 17, pp. 37–54, 1996.
36. R. Robbes and M. Lanza, "Characterizing and understanding development sessions," in *Proceedings of ICPC 2007*, 2007, pp. 155–166.
37. V. Kaptelinin and B. A. Nardi, "Acting with technology: Activity theory and interaction design," *First Monday*, vol. 12, no. 4, 2007. [Online]. Available: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1772
38. W. Snipes, E. Murphy-Hill, T. Fritz, M. Vakilian, K. Damevski, A. R. Nair, and D. Shepherd, *Analyzing Software Data*. Morgan Kaufmann, 2015, ch. A Practical Guide to Analyzing IDE Usage Data.
39. J. Cohen, "The earth is round ($p < 0.5$)," *American Psychologist*, vol. 49, no. 12, pp. 997–1003, 1994.
40. T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 492–501.
41. A. Struyf, M. Hubert, and P. Rousseeuw, "Clustering in an object-oriented environment," *Journal of Statistical Software*, vol. 1, no. 4, pp. 1–30, 2 1997.
42. P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.
43. E. J. Keogh, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, pp. 358–386, 2005.
44. R. Van Solingen, E. Berghout, and F. Van Latum, "Interrupts: just a minute never is," *IEEE software*, no. 5, pp. 97–103, 1998.
45. R. Robbes and M. Lanza, "Improving code completion with program history," *Autom. Softw. Eng.*, vol. 17, no. 2, pp. 181–212, 2010.
46. K. O. McGraw and S. P. Wong, "A Common Language Effect Size Statistic," *Psychological Bulletin*, vol. 111, pp. 361–365, 1992.
47. R. Robbes, D. Röthlisberger, and É. Tanter, "Extensions during software evolution: Do objects meet their promise?" in *Proceedings of ECOOP 2012*, 2012, pp. 28–52.

DOI: 10.1002/smr