

An Empirical Study of the Copy and Paste Behavior during Development

Tarek M. Ahmed, Weiyi Shang, Ahmed E. Hassan
School of Computing
Queen's University
ON, Canada
Email: {tahmed, swy, ahmed}@cs.queensu.ca

Abstract—Developers frequently employ Copy and Paste. However, little is known about the copy and paste behavior during development. To better understand the copy and paste behavior, automated approaches are proposed to identify cloned code. However, such automated approaches can only identify the location of the code that has been copied and pasted, but little is known about the context of the copy and paste. On the other hand, prior research studying actual copy and paste behavior is based on a small number of users in an experimental setup.

In this paper, we study the behavior of developers copying and pasting code while using the Eclipse IDE. We mine the usage data of over 20,000 Eclipse users. We aim to explore the different patterns of Copy and Paste (C&P) that are used by Eclipse users during development. We compare such usage patterns to the regular users' usage of copy and paste during non-development tasks reported in earlier studies. Our findings instruct builders of future IDEs. We find that developers' C&P behavior is considerably different from the behavior of regular users. For example, developers tend to perform more frequent C&P in the same file contrary to regular users, who tend to perform C&P across different windows. Moreover, we find that C&P across different programming languages is a common behavior as we extracted more than 75,000 C&P incidents across different programming languages. Such a finding highlights the need for clone detection techniques that can detect code clones across different programming languages.

I. INTRODUCTION

Copy and Paste (C&P) is a well known practice that is performed by developers in various documents and for different purposes [10, 15]. Developers frequently C&P text during development either by copying their own text or by copying text from external sources. Developers may copy a single variable name, a whole method, an entire class or even an entire file.

Although C&P is a simple and basic action performed by developers, it is believed that C&P is the main cause of code cloning [10, 20]. Code cloning, the process of creating duplicates of existing code, has gained attention in recent years with several studies examining its benefits and harmful side effects. Prior studies find that there is a relationship between less maintainable or defect-prone code, and code cloning [8, 9, 11, 18, 21, 26]. On the other hand, several studies argue that code cloning may not be harmful. Developers may well know the existence of code clones [4], and they may use code clones as an effective design option [13, 15].

To study C&P, automated clone detection techniques are employed in order to discover duplications across the source code. These automated techniques only detect large duplicate code fragments after they are copied and pasted [2, 3, 12, 16, 26]. However the context of such C&P, e.g., how are clones propagated, cannot be detected using such automated clone detection techniques.

To the best of our knowledge, there exists no large scale empirical study on the C&P behavior of real users within a software development IDE. Kim et al [15] explore the C&P behavior of developers by examining the C&P behavior of nine expert developers at IBM T. J. Watson Research Center. The study was based on a small number of research-oriented developers and a short observation period, thus it may not reflect the common behavior of industrial developers.

In this paper, we perform a large empirical study on the C&P behavior within the Eclipse IDE by mining the Eclipse Usage Data Collector (UDC) dataset [1]. UDC is an Eclipse extension that records the user interactions including all used views and editors in addition to all executed commands within the Eclipse IDE. The availability of the UDC data enables us to perform a large scale study of the C&P behaviour. In particular, we analyze the Eclipse UDC dataset for 20,000 users over 20 months, in order to answer the following two research questions:

RQ1: Do IDE users follow the same C&P patterns as regular users?

We find noticeable differences between IDE users and regular users across all C&P patterns. For example IDE users tend to perform more frequent C&P within the same file in contrast to regular users who tend to C&P across different windows. Future empirical studies should explore within file C&P and its impact on code quality.

RQ2: How do IDE users copy and paste code across different file formats?

Our results show that C&P across different file formats is frequent. Hence, techniques are needed to detect clones among different file formats. Such results suggest that automated clone detection techniques and code cloning research should also consider code clones across different languages (e.g., Javascript and PHP).

The rest of the paper is organized as follows: Section II overviews the recent related work. Section III explains the setup of our case study. In section IV we show the results of our two RQs. Sections V discusses Cut&Paste behavior during development. Section VI discusses the threats to the validity of our study. Finally, section VII concludes the paper.

II. RELATED WORK

A. Studies of C&P Behavior

Studies of C&P behavior are performed on a small number of software developers or the studies are done within a non-software engineering context. Kim et al. [15] perform an ethnographic study of Eclipse developers. Kim et al. explore how and why developers tend to C&P code. Two approaches were used in their study, first, developers are observed during development and their C&P actions are manually recorded. Second, the behavior is studied by using a specially-developed tool to capture developers usage of different Eclipse commands. This tool captures two forms of information, the commands executed by the developer, and the changes done to each document corresponding to the recorded command. Their results show some interesting findings about the usage of C&P, Kim et al. report that some limitations in programming languages make C&P unavoidable. For example, the lack of multiple inheritance in Java would require the developer to repeat some code segments. Moreover, they find that developers usually copy code snippets that contain structural templates so they can reuse the templates. Although the study revealed several findings on developers C&P behavior, the study was performed on a small number of experienced software developers in a research center, thus the results may not be applicable to large scale projects or projects including developers of different experience levels. This study and our results complement each other because our results show a more generalized view on the C&P behavior of IDE users whereas the earlier study has a more detailed data of how Eclipse users perform C&P.

Stolee et al. [22] study C&P behavior within a non-software engineering context. They explore the ways that normal users use the clipboard by automatically monitoring clipboard usage for groups of users including administrators, teachers and students. The captured data includes copies, cuts and pastes within the same window and across different windows. Additionally, Stolee et al.'s work defines a set of patterns that describe the C&P behavior. The motivation behind this work is to understand how the C&P behavior can negatively impact the productivity of users by the continuous switching of windows to move text. Finally, Stolee et al.'s work proposes new clipboard strategies to overcome the limitations of traditional C&P tools. In this paper, we examine whether C&P within a software engineering context matches the observations of Stolee et al.'s work.

Murphy-Hill et al. [19] mine the Eclipse UDC dataset as well as three additional datasets to analyze user behavior in the scope of code refactoring. Although the study was not related to C&P directly, it revealed several facts about the behavior

of developers using refactoring commands that are related to the use of C&P. They compare the usage behavior of the developers of Eclipse refactoring commands to the behavior of the users of these commands by analyzing the usage of a large number of refactoring commands including "Rename", "Extract Method" and "Move". They address common assumptions about refactoring tools usage like that developers repeat refactorings, and that they do not usually configure the refactoring commands. In our paper, we mine the same Eclipse UDC dataset, but we focus on C&P commands rather than refactoring commands.

Finally, Yoon and Myers [27] propose the FLUORITE tool to capture low level events in Eclipse. The tool captures additional information about the recorded user interactions than the UDC extension. For example, the Eclipse UDC dataset does not capture moving within the code using the arrow keys or deleting text using the backspace key. More importantly, the FLUORITE captures the content of the executed action, for example, it captures the text involved in each action like the text deleted using the backspace key. Although such information is very useful, the tool has a limited deployment. Hence our study mines the Eclipse UDC dataset that records the interactions of thousands of developers.

B. Studies of Code Cloning

Although there exists a plethora of work done in the detection of code clones [17, 20], **little effort has been devoted to understand the main mechanism for creating clones, i.e., C&P.** Prior studies of code cloning can be primarily categorized along two broad goals: 1) studying the benefits and the harmful effects of code cloning, and 2) proposing new techniques for clone detection, management and removal. **Most of the studies in both categories focus on the post-clone event (caused by a C&P) with minimal explanation C&P process itself.**

Studies on the effects of code cloning primarily used to focus on its harmful side effects [8, 11, 18, 26]. These studies suggest that developers should avoid creating code clones due to their adverse effect on software maintainability and code quality. For example, when a developer clones buggy code, the bug is propagated across different code fragments which makes the bug fixing process more difficult. Fowler [8] considers *duplicated code* as the primary source of bad smells in code. He describes situations where code refactoring is required to decrease the amount of code duplicates caused by C&P.

On the other hand, a few recent studies discuss the benefits of code cloning and consider it as a useful software engineering practice [6, 13, 14]. Kapsler and Godfrey [13] examine the non-harmful side of code cloning by exploring the useful patterns of creating duplicates. Their study presents several situations where code cloning is a desirable practice rather than a harmful one. One of these situations is *Forking* where a developer intends that the cloned and original code fragments would evolve independently.

Some studies propose tools for code clones management. Code clones management is a way to keep track of the cloned code to provide details about the clones to the IDE users. Duala-Ekoko and Robillard [7] propose CloneTracker, a tool to manage clones in the Eclipse IDE. The input to CloneTracker is the output of SimScan tool, a tool to detect code clones in Eclipse. CloneTracker tracks the occurrence and evolution of code clones and have the ability to inform developers when changes are applied to cloned regions. Similarly, Chiu and Hirtle [5] propose Clonescape tool. Clonescape is an Eclipse plugin that tracks code clones. Unlike CloneTracker, Clonescape does not require the input from SimScan tool, Clonescape can automatically tag C&P and mark them as code clones. Toomim et al. [25] propose a Linked Editing technique implemented in CodeLink prototype editor. The technique requires that the developer manually links the cloned code to the original code. The technique then calculates the differences and visualizes them so that the developer is aware of all the changes in the cloned code.

Our work focuses on the C&P behavior of Eclipse IDE users in order to gain more insight about this practice in Eclipse.

III. CASE STUDY SETUP

In this section we describe the Eclipse Usage Data Collector (UDC) dataset. We present how this dataset captures the behavior of Eclipse IDE users and we show the setup of our study that explored the C&P behavior of the Eclipse IDE users.

A. Eclipse Usage Data Collector (UDC)

In this paper, we use the Eclipse Usage Data Collector (UDC) dataset [1]. UDC is an extension that captures how users use the Eclipse IDE by recording all performed actions. This data can be used to help researchers to understand the behavior of Eclipse users. We mine the UDC dataset that has been collected between January 2009 to August 2010. Typically, UDC captures four types of information:

- 1) Plug-ins: Information about all installed Eclipse plugins that are started with the IDE.
- 2) Application commands: Information about the actions done by the users. The actions include shortcut keys and actions performed using menu items.
- 3) Changes in Perspectives: Information about the used Eclipse perspectives like Java perspective or Debug perspective.
- 4) Editor events: Information about when a user opens, closes or gives focus to a specific editor. For example, UDC would record when the Java or C editors are activated.

Table I shows an example of the UDC data where a developer activates a Java editor, performs a C&P and finally saves the file. Each record consists of: 1) a unique user ID, 2) the performed action (like activation or execution), 3) what the action is performed on (like editor or command), 4) the Eclipse bundle responsible for the action, 5) the bundle version, 6) a description of the action, showing more details about the

performed action, and 7) the time when the action is performed (in milliseconds).

B. Data Pre-Processing

In this sub-section, we discuss our data pre-processing. Our pre-processing consists of four steps: 1) Creating Development Sessions, 2) Finding Active Sessions, 3) Finding Frequent Users, and 4) Finding C&P.

1) *Creating Development Sessions*: First, we define the concept of a development session. A development session is calculated by extracting the commands of a specific user until one of two conditions is encountered: either the user closes the Eclipse IDE, or an idle duration of at least two hours is found between two consecutive records. We revisit this limit in Section V.

2) *Finding active sessions*: Second, we find the active development sessions. A development session can be very short and not meaningful because the user may turn off the UDC extension or may open the IDE without considerable activity. We define the notion of an active session where a session is considered active if it has more than 10 commands of any type. We only consider a session active if it has more than 10 commands such that we have a more realistic view about the users' behavior by eliminating sessions with very small number of user actions. We note that the sessions having less than 10 commands have 0.346 C&P incidents on average and standard deviation of 0.79. Therefore we decide to exclude these sessions. Our data has 7,653,647 active sessions for 826,609 users. The median number of sessions for all users in the dataset is five sessions, the average is 9.25 sessions and the standard deviation is 13.26 sessions.

3) *Finding frequent users*: Third, we find users who use the IDE frequently. We consider a user having more than 50 hours of IDE activity as a frequent user. The motivation behind selecting users with more than 50 hours is that we compare C&P behavior in the Eclipse UDC dataset with results of regular users as reported in an earlier study, in which the authors report the results for users with an average of 50 hours of observation [22]. Our data has 21,770 users with more than 50 hours of activity. These users have a median of 70 hours of activity and a standard deviation of 35 hours. Table II shows the five-number summary and the mean value of the hours of activity of all users.

4) *Finding C&P*: Finally, we extract all C&P incidents performed by the set of active and frequent users. During the extraction process, we record the currently used editor type and the time that this editor was activated in order to keep track of the different files and editors that the developer copies from and pastes into.

TABLE II
FIVE-NUMBER SUMMARY AND THE MEAN VALUE OF THE HOURS OF ACTIVITY OF ALL USERS WITH MINIMAL 50 ACTIVE HOURS

Min	1st Qu.	Median	Mean	3rd Qu.	Max
50	58.22	70.6	82.11	93.38	684.4

TABLE I
EXAMPLE OF UDC DATASET

User ID	What	Kind	Bundle	Bundle Version	Description	Timestamp
382620	activated	editor	org.eclipse.jdt.ui	3.4.0.v20080603-2000	org.eclipse.jdt.ui.CompilationUnitEditor	1246397908385
382620	executed	command	org.eclipse.ui	3.4.0.I20080610-1200	org.eclipse.ui.edit.copy	1246397917867
382620	executed	command	org.eclipse.ui	3.4.0.I20080610-1200	org.eclipse.ui.edit.paste	1246397920110
382620	executed	command	org.eclipse.ui	3.4.0.I20080610-1200	org.eclipse.ui.file.save	1246397920846

We extracted more than 4 million C&P incidents. A C&P incident is defined as a copy followed by one or more pastes. We also capture a special type of paste incident where a user activates the IDE followed by a paste command which means the user may have copied from an external source and pasted into the Eclipse IDE. We find that 23.96% of the C&P incidents consists of pastes from external sources.

We find that the average number of C&P incidents is 2.72 per hour, while as reported in earlier studies [15, 22], 3 to 4 C&P incidents per hour are performed by regular users and 16 C&P incidents per hour are performed by experienced Eclipse users. Table III shows the 5-number summary+mean value of the C&P incidents per user per hour.

TABLE III
FIVE-NUMBER SUMMARY AND THE MEAN VALUE OF CLIPBOARD INTERACTIONS PER HOUR

Min	1st Qu.	Median	Mean	3rd Qu.	Max
0.004	1.122	2.1	2.72	3.65	21.43

IV. CASE STUDY RESULTS

RQ1: DO IDE USERS FOLLOW THE SAME C&P PATTERNS AS REGULAR USERS?

Motivation. The motivation behind this RQ is that Eclipse IDE does not provide more support for C&P other than the very basic C&P functionality. Therefore, we study whether the C&P behavior of IDE users is the same as regular users in order to explore the need for special C&P support tools in the Eclipse IDE. We perform a study on the C&P behavior of IDE users where we explore the different patterns of usage and compare the C&P behavior of IDE users to that of regular users. We compare our results with the results reported in a previous study that studies the C&P behavior of regular computer users [22].

Approach. A prior study of the C&P behavior for regular users defines two different types of patterns: Elementary Patterns and Complex Patterns [22]. The elementary patterns are composed of a single C&P interaction involving one or more files. On the other hand, complex patterns are composed of two or more C&P incidents involving more than two files.

The elementary patterns are broken into two sub-types: “*Between*” and “*Within*”.

- **Between:** A pattern that involves a C&P incident such that both the copy and the paste commands are performed

in two different windows. This pattern consists of a developer opening a file, performing a copy command, then opening another file and performing a paste command. As the Eclipse UDC dataset does not provide a unique identifier for files, we consider opening an editor window as opening a new file.

- **Within:** A pattern that involves a single file where both C&P commands are executed. This pattern consists of a developer opening a file, performing a copy command then performing one or more paste commands in the same file.

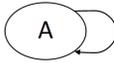
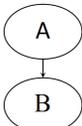
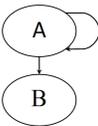
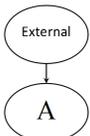
In addition to the two previously identified patterns for regular users, we propose two new patterns:

- **Within and between:** A pattern that captures an incident where the code is pasted both in the same file from which it was copied, in addition to one or more other files.
- **External paste:** A pattern that involves an IDE user activating the Eclipse IDE followed by one or more pastes without performing any other C&P incidents.

We present below the definitions of the five sub-types of complex patterns as mentioned in the previous study [22] and discuss whether the available UDC dataset can be used to detect each of these pattern.

- **Repeat:** A repeat pattern is defined as two or more consecutive C&P incidents such that both the source and destination windows are the same in all incidents. The Eclipse UDC dataset does not provide a unique identifier for each file, thus we assume that a set of C&P incidents that are performed during a short period of time can be considered as a repeat pattern. We detect two or more consecutive patterns having the following sequence [Activate editor - Copy - Activate editor - Paste] in a short period of time (two minutes) as a repeat incident. We choose two minutes heuristically because we are unable to identify exactly how the IDE user repeats C&P.
- **Distribution:** A distribution pattern involves a copy from a common source file and two or more pastes in different distinct files. The Eclipse UDC dataset does not guarantee the uniqueness of the editor being pasted into, so a distribution pattern in this case can be detected when a developer first activates an editor and performs a copy command, then repeat a sequence of [Activate editor - Paste] without performing any other clipboard interactions.
- **Composition:** A composition pattern is defined as two or

TABLE IV
C&P ELEMENTARY PATTERNS. μ IS THE AVERAGE PERCENTAGE OF EACH PATTERN AND σ IS THE STANDARD DEVIATION

Pattern	Definition	C&P Usage For IDE users		C&P Usage for End-Users	
		Per User	Overall	Per User	Overall
Within File 	A pattern involving a C&P within the same file	$\mu = 60.26\%$ $\sigma = 15.89\%$	63.52%	$\mu = 23\%$ $\sigma = 18\%$	35%
Between Files 	A pattern involving a copy and one or more pastes in a different file	$\mu = 15.35\%$ $\sigma = 9.55\%$	15.39%	$\mu = 77\%$ $\sigma = 18\%$	65%
Within and Between Files 	A pattern involving a copy followed by two or more pastes both in the same file and in different files	$\mu = 0.42\%$ $\sigma = 0.88\%$	0.39%	N/A	N/A
External Paste 	A pattern involved by a user activating Eclipse IDE followed by a paste command directly	$\mu = 23.96\%$ $\sigma = 14.8\%$	20.69%	N/A	N/A

more C&P incidents with text that is copied from multiple files and pasted into a single destination file. This pattern cannot be detected using the Eclipse UDC dataset because there is no unique identifier for each file.

- **Isolation:** A pattern that involves an incident where the source and destination files are different from the files that are involved in the previous or following incident. This pattern cannot be detected because there is no way to determine whether the files that are involved in the incident are different from the other incidents (due to privacy protection of UDC users).
- **Relay:** This pattern involves two consecutive incidents where the destination of the first incident is the source of the second incident. This pattern can be detected when a developer first activates an editor, performs a copy command, then activates an editor, performs a paste followed by a copy, and finally opens another editor and pastes in it. This sequence can be summarized as [Activate Editor - Copy - Activate Editor - Paste - Copy - Activate Editor - Paste]. However the original definition is that all the windows are different, which is not detectable using UDC dataset [22].

In particular, we explore three complex patterns: *Repeat*, *Distribution* and *Relay* in addition to an extra *Unknown* pattern

where a C&P incident does not belong to any of the other complex patterns. Therefore, a C&P incident is classified as *Unknown* if it does not have a relation to either its previous or next C&P incidents.

Results. We extract 4,058,046 incidents, 3,218,437 (76.04%) of which are internal C&P incidents and 839,609 are external paste incidents.

Elementary patterns analysis shows a noticeable difference between the behavior of IDE users and regular users. We observe that IDE users tend to more frequently perform C&P in the same file rather than across different files. We find that 60.62% of the incidents belong to “*Within*” pattern compared to 23% for regular users. Moreover, the percentage of “*Between*” pattern is 15.35% compared to 77% for regular users. Table IV shows the percentages of occurrence of elementary patterns relative to the total number of incidents. The table shows two types of results in *C&P usage* column. First *Per User* results, we calculate the percentages of incidents for each user and we report the mean and standard deviation for these percentages. Second *Overall* results is the percentage of occurrence across the whole dataset. Moreover the table shows a comparison with the behavior of regular users except for “*Within and Between*” and “*External Paste*” patterns which were not reported for regular users.

TABLE V
C&P COMPLEX PATTERNS. μ IS THE AVERAGE PERCENTAGE OF EACH PATTERN AND σ IS THE STANDARD DEVIATION

Pattern	Description	C&P Usage For IDE Users		C&P Usage For End-Users	
		Per User	Overall	Per User	Overall
Repeat	A pattern formed when a user activates an editor and copies then opens another editor and pastes and repeat this operation more than once in a short period of time (less than two minutes)	$\mu = 8.94\%$ $\sigma = 6.2\%$	10.95%	$\mu = 32\%$ $\sigma = 18\%$	37%
Distribution	A pattern formed when a user activates an editor and perform a copy command then perform paste command in two or more different files	$\mu = 0.82\%$ $\sigma = 1.28\%$	0.7%	$\mu = 36\%$ $\sigma = 16\%$	32%
Relay	A pattern formed when a user activates an editor, performs a copy then opens another editor and pastes. Finally the user copies from the same editor and pastes in a third editor	$\mu = 32.94\%$ $\sigma = 15.65\%$	39.43%	$\mu = 2\%$ $\sigma = 2\%$	3%
Unknown	A single C&P interaction that doesn't fit in any of the above patterns	$\mu = 57.29\%$ $\sigma = 16.85\%$	48.92%	N/A	N/A

Complex pattern analysis shows major differences between C&P behavior of IDE users and regular users. For example a repeat pattern is found on average in 8.94% of the incidents compared to 32% for regular users. A distribution pattern is much less used by IDE users where we only find 0.82% of the incidents where a user pastes in two or more different files. Finally, a relay pattern seems to be used frequently as an IDE user pastes some code in a specific file then copies from the same file and pastes in other files. Table V shows the percentages of occurrence of each complex pattern relative to the overall number of incidents.

One difference between IDE users and regular users is that IDE users perform more relay patterns while regular users perform more distribution patterns. The reason behind this phenomenon is that the copied text needs more edit in case of source code since code usually does not work if it is just copied to another location. The relay pattern gives IDE users a chance to edit the code unlike the distribution pattern which the pasted code is not edited.

The relay pattern appears frequently for IDE users. One explanation of the frequent appearance of a relay pattern is after a developer clones a method or variable, this method or variable is then copied and pasted to be involved or referenced in other locations in the source code. To better understand how the relay pattern affects the code, we calculate the difference in duration between the end of the first C&P incident and the start of the second C&P incident involved in the same relay pattern. We find that the median duration is 20.86 seconds, the average duration is 126.64 seconds and the third quartile of the duration is 64.16 seconds. This implies that although IDE users relay their code frequently, there is a small time difference within the relay incident that does not give the developer a chance to make major code changes.

Additionally we notice a small frequency of the repeat pattern. Our detected repeat patterns are an over estimation of the actual repeat patterns because of the absence of the file identifier. If there is a file identifier for each opened file, the detected repeat patterns will be fewer. We explain the small percentage of repeat pattern by the fact that information in

the code is less scattered than other sources. For example if a developer is willing to copy line 10 to line 20 and line 50 to line 60, he may just copy line 10 to 60 and remove the unwanted lines in the pasted file.

The unknown pattern includes C&P incidents that do not belong to any of the other complex patterns. However, we investigated these incidents in order to extract other patterns of interest. One of the patterns that we observed is “within-file repeat”, where the IDE user opens a file, performs a copy command and follows it with multiple paste commands in the same file. We could not find a specific rationalization for such a pattern thus we decided to only report it as one of the unknown patterns.

Our results can be used to improve C&P support in Eclipse. For example, Eclipse can track the source and the multiple destinations of each C&P incident. Therefore, IDE users can perform the distribution and relay patterns without worrying about forgetting where they have pasted.

In summary, unlike regular users, IDE users tend to use C&P in a different manner. Therefore the C&P support tools for regular users may not be as beneficial for IDE users [23].

The C&P behavior of Eclipse IDE users is different from the behavior of regular users. Accordingly, Eclipse IDE requires tailored C&P support tools that differ from regular users' C&P tools. For example, because of the large number of relay pattern, users may need to know where they have pasted. Therefore IDEs should track the C&P incident source and destinations

RQ2: HOW DO IDE USERS COPY AND PASTE CODE ACROSS DIFFERENT FILE FORMATS?

Motivation. IDEs (like Eclipse) typically support different types of editors. IDE users are able to copy and paste text across different editors. However, clone detection techniques often do not consider code clones between different languages [20]. We want to investigate whether IDE users copy and paste text across different editors and across which editors do IDE users copy and paste text.

Approach. In order to answer this RQ, first we extract all the editor types that are recorded in the Eclipse UDC dataset. Some editor types correspond to a programming language like Java, C or Python. Other editor types can be used across languages and for different purposes like the XML, Text and Compare editors. Second, we record the current opened editor type for each clipboard interaction. For example, a user activates a Java editor, and performs a copy command, then he/she activates a JSP Editor and performs a paste command. We record that the source of this incident is a Java editor and the destination is a JSP Editor. Table VI shows an example of a developer who copies from a Java editor and pastes into a JSP Editor.

We classify C&P incidents into three patterns: “*Within Editor*”, “*Between Editors*” and “*Within and between Editors*”.

- **Within Editor:** One C&P incident includes both the copy and the paste commands in the same type of editor.
- **Between Editors:** A copy is performed in editor A and the paste is performed in a different type of editor B.
- **Within and between Editors:** A developer copies from editor A then performs two or more pastes in the same editor type A and in a different type of editor B.

In addition, we examine which editors are more frequently used to perform C&P between and within editors, respectively.

Results. IDE users usually use multiple types of editors. The Eclipse UDC dataset contains 18 types of editors. On average a developer uses three different editor types (with some users using as much as 12 editors).

Most of the C&P are in the same type of editors. Among all the 3,218,437 C&P incidents, we find that more than 97% of these incidents are within the same editor while 2.36% of the incidents are between different editors (shown in Table VII). Although the percentage of “*Between Editor*” pattern is low, there are still more than 75,000 C&P incidents. Hence we decide to explore the different patterns of C&P occurrences for these incidents.

TABLE VII
C&P USAGE IN EDITOR TYPES. μ IS THE AVERAGE PERCENTAGE OF EACH PATTERN AND σ IS THE STANDARD DEVIATION

Pattern	Definition	C&P Usage	
		Per User	Overall
Within Editor	A pattern involving a C&P within the same editor type	$\mu = 97.55\%$ $\sigma = 4.15\%$	97.62%
Between Editors	A pattern involving a copy and one or more pastes in a different editor types	$\mu = 2.36\%$ $\sigma = 4.08\%$	2.31%
Within and Between Editors	A pattern involving a copy followed by two or more pastes both in the same editor type and in different editor types	$\mu = 0.07\%$ $\sigma = 0.4\%$	0.06%

There is a large number of C&P between Java and XML editors, as well as Java and JSP editors. Despite that both languages have a completely different syntax, the XML editor is the destination of 33.62% of the copy instances from the Java editor. This can be explained by the fact that there is a large usage of XML in Java frameworks (e.g., Spring¹). IDE users may share code between XML and Java editors by C&P. Moreover, IDE users may copy Java code from a Java class and paste it into a JSP page since the JSP editor is the destination for 22.55% of the incidents where the source is a Java editor. For example, a developer can duplicate some code that validates an input from a Java class to a JSP page, the validation may be on a date format or email format or any other form of validation. The duplication makes the code harder to maintain as the application evolves, thus, clone detection techniques need to handle this type of clones.

There is a considerable amount of C&P across editors that are used in web development. For example, the PHP editor has many C&P with the Javascript editor (21.38%) and the HTML editor (13.52%). Code used in web development can exist interchangeably in several file types. The interchangeable code may increase the effort needed to maintain such web applications. Additionally, the interchangeable code would increase the expense of maintenance of the web application. For example an HTML developer needs to know Javascript as well, to be able to maintain the application.

The C language editor has the least percentage of “*Across Editors*” incidents relative to the total number of incidents where the C editor is the source of the C&P. Additionally, the destination of these incidents is a Text editor in more than 65% of the incidents and is a Java editor in 11% of the incidents. Therefore, the interaction between the C language and other editor types is minimal because of the nature of the C language itself which is primarily used to develop standalone applications that do not require text to be moved across different editor types. On the other hand, Java is heavily used in web applications development that require moving text across different editors.

Although C&P across different editors has a small percentage compared to C&P within the same editor type. This behavior may result in code clones across different languages highlighting the need for clone detection techniques to detect clones across different languages.

IDE users may use C&P across different editors. Clone detection techniques should also consider detect clones across different languages.

V. DISCUSSION

In this section we further discuss our results, first by extending our analysis to Cut&Paste as a tool of code movement. Second, we compare our results of the average rate of C&P per hour to what is reported in earlier studies.

¹<http://projects.spring.io/spring-framework/>

TABLE VI
EXAMPLE OF A DEVELOPER PERFORMING C&P IN DIFFERENT EDITORS

User ID	What	Kind	Bundle	Bundle Version	Description	Timestamp
104526	activated	editor	org.eclipse.jdt.ui	3.4.0.v20080603-2000	org.eclipse.jdt.ui.CompilationUnitEditor	1231303687537
104526	executed	command	org.eclipse.ui	3.4.0.I20080610-1200	org.eclipse.ui.edit.copy	1231303691532
104526	activated	editor	org.eclipse.jst.jsp.ui	1.1.300.v200805152207	org.eclipse.jst.jsp.core.jspsource.source	1231303694442
104526	executed	command	org.eclipse.ui	3.4.0.I20080610-1200	org.eclipse.ui.edit.paste	1231303706041

TABLE VIII
INTERACTION BETWEEN DIFFERENT EDITORS

	% Destination									% Across Editor	
	Java	PHP	C	JSP	XML	Javascript	HTML	Text	Others		
Source	Java	-	0.1424	0.3165	22.5545	33.6182	2.0950	2.9526	24.1147	14.2061	1.56
	PHP	0.6585	-	0.0760	0.2026	3.1278	21.3752	13.5241	37.6472	23.3886	1.23
	C	11.2431	0.2375	-	0	5.7007	0.7918	0.3167	65.3207	16.3895	0.39
	JSP	46.2807	0.0512	0	-	9.7848	19.0779	3.5143	1.7418	19.5492	5.98
	XML	63.6494	3.2680	0.6446	8.8817	-	1.8892	1.6385	5.7660	14.2627	8.44
	Javascript	9.3380	28.1894	0.0219	29.1977	3.6826	-	16.6813	4.1210	8.7681	4.89
	HTML	12.0815	28.9283	0.0354	10.1151	3.1532	20.9212	-	4.2161	20.5492	8.63
	Text	36.4180	21.1442	7.4467	3.7097	9.6073	4.7968	4.0766	-	12.8007	6.46

A. Code Movement (Cut&Paste)

We further extend our analysis to compare the usage of code movement (Cut&P) to code cloning (C&P). We perform the same analysis with Cut command rather than Copy. We find 623,369 Cut&P incidents, which is around 85% less than the number C&P incidents. We notice that the percentage of Cut&P within the same file is more than 90%, which indicates that moving code is a common practice within the same file rather than between different files. Table IX shows the Cut&P patterns identified with the Eclipse UDC dataset. There is a considerable difference between the results of code movement and code cloning tools where IDE users cut and paste in the same file more frequently than C&P (93% compared to 60.62% for C&P).

We also repeat the analysis of the complex patterns in Table X. We notice that the average percentage of the Cut&P repeat pattern is higher than the C&P repeat pattern. However, the other Cut&P patterns are in the same range as the C&P.

As discussed in RQ1, IDE users do not perform a large number of C&P repeat patterns. However, in the case of Cut&P, if a developer is willing to move lines 10 to 20 and lines 50 to 60 from file A to file B, the developer typically would perform this process in two steps: first moving lines 10 to 20, then moving lines 50 to 60, in order to not delete lines 20 to 50 from file A.

B. Difference between our results and earlier studies

An earlier study of the C&P behavior of experienced Eclipse users reported an average 16 C&P incidents per hour [15], on the other hand, our study reports an average of 2.73 C&P incidents per hour. The difference of our results compared

TABLE IX
CUT & PASTE ELEMENTARY PATTERNS. μ IS THE AVERAGE PERCENTAGE OF EACH PATTERN AND σ IS THE STANDARD DEVIATION

Pattern	Definition	Cut&P Usage	
		Per User	Overall
Within File	A pattern involving a cut and paste within the same file	$\mu = 93.57\%$ $\sigma = 10.53\%$	93.68%
Between Files	A pattern involving a cut and one or more pastes in a different file	$\mu = 6.02\%$ $\sigma = 10.26\%$	15.39%
Within and Between Files	A pattern involving a cut followed by two or more pastes both in the same file and in different files	$\mu = 0.02\%$ $\sigma = 2.32\%$	0.34%

to other studies is because we perform a large scale study on a large number of users of different levels of expertise. Moreover, the choice of our threshold of session idle time of 2 hours may impact the rate of C&P.

In order to further explore our result, we perform an analysis on the rate of C&P in heavy editing sessions. We define a heavy editing session as a session having a total number of commands exceeding the average number of commands per session (μ) by a multiple of the standard deviation of the number of commands per session (σ). Additionally, we modify the threshold of two hours to be ten minutes and perform the heavy editing analysis. We show the results in Table XI. The results show that when we minimize the idle time in the heavy

TABLE X
CUT&P COMPLEX PATTERNS. μ IS THE AVERAGE PERCENTAGE OF EACH PATTERN AND σ IS THE STANDARD DEVIATION

Pattern	Description	Cut&P Usage	
		Per User	Overall
Repeat	A pattern formed when a user activates an editor and cuts then opens another editor and pastes and repeat this operation more than once in a short period of time (less than two minutes)	$\mu = 16.02\%$ $\sigma = 13.14\%$	21.4%
Distribution	A pattern formed when a user activates an editor and perform a cut command then perform paste command in two or more different files	$\mu = 0.27\%$ $\sigma = 2.45\%$	0.18%
Relay	A pattern formed when a developer user an editor, performs a cut then opens another editor and pastes. Finally the user cuts from the same editor and pastes in a third editor	$\mu = 26.99\%$ $\sigma = 16.86\%$	33.68%
Unknown	A single Cut&P interaction that doesn't fit in any of the above patterns	$\mu = 56.71\%$ $\sigma = 21.01\%$	44.74%

editing sessions, we obtain a C&P rate close to the results reported in the earlier study [15].

TABLE XI
ANALYSIS OF THE RATE OF C&P PER HOUR

	Average rate of C&P per hour for heavy editing sessions	
	$\mu + \sigma$	$\mu + 2\sigma$
Sessions with less than two hours idle time	7.6 C&P/hour	8.88 C&P/hour
Sessions with less than ten minutes idle time	11.39 C&P/hour	13.18 C&P/hour

C. Challenges of studying C&P behavior of IDE users

More studies are needed to fully understand the C&P behavior of IDE users on a large scale. However, such studies may face several challenges. The first challenge is the privacy of the recorded data; for example, the Eclipse UDC dataset does not track two important sources of information, the file identifiers and the content of the event. The creation of one way hashes for file identifiers and event contents and the tracking of such hashes might be a middle ground to enable more elaborate large scale studies while ensuring the privacy of Eclipse users.

Terry et al. [24] explore a similar approach on an open source software for image manipulation (GIMP). In particular, Terry et al. examine a version of GIMP software called Instrumental GIMP (ingimp). This version contains a UDC-like extension that records the users' usage information. The recorded data includes a more detailed usage information than the Eclipse UDC dataset, including file identifiers and information about the used images. Hence, the authors discuss the recorded information and how they tackle the privacy concerns of such information. For example, ingimp records whether the user works on the same file or different files. Therefore, a file identifier has to be recorded. Ingimp generates an arbitrary 32-bit number for each file name in addition to a 32-bit hash associated to the number. The association between the hash and the number is stored on the user's machine,

and only the arbitrary number is recorded in the log file. This allows the data to contain distinct file identifiers while preventing other third party software from reconstructing the original file name.

The second challenge is the performance of the recording tool and its effect on the usability of the IDE. Capturing more data leads to more processing on the user's machine. In a large scale study it is more likely that users with computers of different capabilities use the IDE. Therefore the recording tool should be able to capture data with the minimal performance overhead.

VI. LIMITATIONS AND THREATS TO VALIDITY

In this section we discuss the threats to validity of our study.

A. Internal Validity

In this study, we perform a large scale analysis of thousands of Eclipse users performing C&P. We analyze the usage of C&P in different files and in different editors. We compare the C&P patterns of IDE users to C&P patterns of regular users, however both studies are not directly comparable. The study of regular users was done on the window level, so if a specific application has an internal tabbing functionality then we would not get any idea how C&P is done within this application. Hence, it is essential for us to look into the Eclipse UDC dataset to get an IDE focused view of C&P.

B. External Validity

In this paper, we study the Eclipse UDC dataset only. Although Eclipse is a widely used IDE by millions of users, our results may not generalize to other popular IDEs like IntelliJ and Netbeans.

Moreover, Eclipse is used by users of different programming languages, however, its main focus is on Java. We tried to extract information about the usage of C&P in different programming languages in Eclipse but due to the small scale of usage, our results may not generalize to other programming languages.

The Eclipse UDC dataset was collected during the period from January 2009 to August 2010. This is the latest publicly

available snapshot of the UDC dataset. The practice of C&P may have changed since the collection of the dataset, therefore, in case of the availability of another dataset containing the same type of information, more research may be done to verify our results.

C. Construct Validity

Although the Eclipse UDC dataset is a rich dataset and contains valuable information about the usage of the IDE, however, the Eclipse UDC dataset has drawbacks that may cause some limitations in our analysis. In the following subsections, we categorize the limitations of our analysis and show the impact of such limitations on our results.

1) *Assumptions*: In this study, we make several assumptions about the data. First, a copy or paste command may mean copying code or copying a whole file. The Eclipse UDC dataset does not provide enough information to differentiate between both actions, to mitigate this risk, we consider copy commands only if the user activates a specific editor to obtain more accurate results. Second, We assume the end of development session either when the user closes the workbench or if we find an idle duration of two hours between two consecutive records, this may not be realistic in case that a user is restarting the IDE. In addition, our data filtration is based on our own analysis of the UDC dataset. Hence adopting a different filtration setup may produce different results.

Due to dataset limitations, some of the most common complex C&P patterns cannot be detected or can be detected by adopting specific assumptions. For example, the isolation pattern cannot be detected because the UDC dataset does not provide any identifier for the opened file, thus, a unique key does not exist for the opened file and there is no way to identify if the files that are involved in a specific C&P incident differ from the files in a previous C&P incident.

2) *Dataset limitations*: In this subsection, we discuss the various UDC dataset limitations.

Missing File identifiers. The Eclipse UDC dataset does not provide a unique identifier for each file. This impacts the validity of our study. Consider the following two usage scenarios: Scenario 1: User opens file f1, performs a copy, opens file f2, then opens file f1 again then performs a paste. In this scenario, we consider this event as “between” C&P although actually it is a “within” event. This leads to an overestimation of the “between” pattern.

Scenario 2: User opens file f1, performs a copy, opens file f2 and pastes, then returns back to f1 and performs another paste. In this scenario, we consider this as a “between” pattern although it is a “within and between” pattern. This may overestimate “between” pattern.

Missing C&P content. The Eclipse UDC dataset does not track the actual content of C&P incident due to privacy concerns. Our analysis is based solely on the incidents of C&P themselves rather than the copied code, which limits our analysis in various ways. For example, we are unable to differentiate between the trivial C&P incidents involving a

variable name or a code comment and the major ones involving a method, a class or a whole file.

Additionally, consider the following scenario: A user opens file f1, performs a copy, then leaves the IDE, he returns to the IDE again and pastes, we consider this as an external paste event because the user is assumed to bring code from an external source as we do not know the event content. This will overestimate external C&P events.

Missing the content of other commands. We find that paste commands are sometimes followed by delete commands, it is not known if the delete commands are related to parts of the pasted code or simply deleting blank lines to format the code. Moreover, we find evidence of undo commands being executed after a paste command in 2.38% of the C&P incidents, however we argue that this percentage does not affect the overall accuracy of our results.

Missing some keyboard commands. UDC does not capture some commands like arrow keys and backspace, this will not affect our analysis of C&P events because the missing commands are not directly related to the C&P commands.

Missing drag and drop commands. UDC does not capture when the IDE user drags code in the same file which acts as a Cut&P incident. This will result in underestimation of within Cut&P events presented in the discussion but it will not affect our C&P analysis.

Table XII shows a summary of the dataset limitations threats and the impact of each threat on the results.

TABLE XII
THREATS IN OUR ANALYSIS AND THEIR IMPACT ON THE RESULTS

Threat	Impact
Missing File Identifiers	Overestimation of between files pattern
Missing C&P content	Overestimation of external C&P incidents
Missing content of other commands	Unable to identify if the user deletes code or blank lines
Missing some keyboard commands	No effect on the results
Missing drag and drop commands	Underestimation of within Cut&P incidents

VII. CONCLUSION

In this paper we perform a large empirical study on the C&P behavior of Eclipse users. We observe several differences between the C&P behavior of IDE users and regular users. IDE users tend to perform C&P within the same file unlike regular users who perform C&P across different files. We also find that IDE users frequently perform C&P across different editor types. Our findings imply that Eclipse IDE requires C&P support tools tailored for IDE users because of the different patterns of usage. Moreover, clone detection techniques should consider locating clones across different file types rather than primarily focusing on clones within the same file type.

REFERENCES

- [1] "Eclipse usage data collector (udc)." [Online]. Available: <https://eclipse.org/epp/usagedata/>
- [2] B. Baker, "On finding duplication and near-duplication in large software systems," in *Proceedings of 2nd Working Conference on Reverse Engineering (WCRE '95)*, Jul 1995, pp. 86–95.
- [3] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings of International Conference on Software Maintenance (ICSM '98)*, Nov 1998, pp. 368–377.
- [4] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. Hassan, "An empirical study on inconsistent changes to code clones at release level," in *Proceedings of 16th Working Conference on Reverse Engineering (WCRE '09)*, Oct 2009, pp. 85–94.
- [5] A. Chiu and D. Hirtle, "Beyond clone detection," Chertton School of Computer Science, University of Waterloo, Apr 2007.
- [6] E. Duala-Ekoko and M. Robillard, "Tracking code clones in evolving software," in *Proceedings of 29th International Conference on Software Engineering (ICSE '07)*, May 2007, pp. 158–167.
- [7] E. Duala-ekoko and M. P. Robillard, "Clonetracker: Tool support for code clone management," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, 2008, pp. 843–846.
- [8] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*, ser. Addison-Wesley Object Technology Series. Pearson Education, 2012.
- [9] A. Hunt and D. Thomas, *The Pragmatic Programmer: from Journeyman to Master*. Addison-Wesley Professional, 2000.
- [10] P. Jablonski, "Managing the copy-and-paste programming practice in modern IDEs," in *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '07)*. New York, NY, USA: ACM, October 2007, pp. 933–934.
- [11] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *Proceedings of IEEE 31st International Conference on Software Engineering (ICSE '09)*, May 2009, pp. 485–495.
- [12] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.
- [13] C. Kapsner and M. Godfrey, "'Cloning Considered Harmful' considered harmful," in *Proceedings of 13th Working Conference on Reverse Engineering, (WCRE '06)*, Oct 2006, pp. 19–28.
- [14] C. J. Kapsner and M. W. Godfrey, "Supporting the analysis of clones in software systems: Research articles," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 2, pp. 61–82, Mar 2006.
- [15] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proceedings of International Symposium on Empirical Software Engineering (ISESE'04)*, 2004, pp. 83–92.
- [16] K. A. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein, "Reverse engineering," L. Wills and P. Newcomb, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 1996, ch. Pattern Matching for Clone and Concept Detection, pp. 77–108.
- [17] R. Koschke, "Identifying and removing software clones," in *Software Evolution*. Springer Berlin Heidelberg, 2008, pp. 15–36.
- [18] A. Lozano and M. Wermelinger, "Assessing the effect of clones on changeability," in *Proceedings of IEEE International Conference on Software Maintenance (ICSM '08)*, Sept 2008, pp. 227–236.
- [19] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," in *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 287–297.
- [20] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470 – 495, 2009, special Issue on Program Comprehension (ICPC '08).
- [21] G. Selim, L. Barbour, W. Shang, B. Adams, A. Hassan, and Y. Zou, "Studying the impact of clones on software defects," in *Proceedings of 17th Working Conference on Reverse Engineering (WCRE '10)*, Oct 2010, pp. 13–21.
- [22] K. T. Stolee, S. Elbaum, and G. Rothermel, "Revealing the copy and paste habits of end users," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '09)*. IEEE, 2009, pp. 59–66.
- [23] J. Stylos, B. A. Myers, and A. Faulring, "Citrine: Providing intelligent copy-and-paste," in *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. Santa Fe, NM, USA: ACM Press, Oct 2004, pp. 185–188.
- [24] M. Terry, M. Kay, B. Van Vugt, B. Slack, and T. Park, "Ingimp: Introducing instrumentation to an end-user open source application," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. New York, NY, USA: ACM, 2008, pp. 607–616.
- [25] M. Toomim, A. Begel, and S. Graham, "Managing duplicated code with linked editing," in *Proceedings of IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC '04)*, Sept 2004, pp. 173–180.
- [26] V. Wahler, D. Seipel, J. W. von Gudenberg, and G. Fischer, "Clone detection in source code by frequent itemset techniques," in *Proceedings of 4th IEEE International Workshop on Source Code Analysis and Manipulation*

(SCAM '04), vol. 4, Chicago, IL, USA, 2004, pp. 128–135.

[27] 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 (PLATEAU '11). New York, NY, USA: ACM, 2011, pp. 25–30.