The Impact of View Histories on Edit Recommendations

Seonah Lee, *Member, IEEE,* Sungwon Kang, *Member, IEEE,*
Sunghun Kim, *Member, IEEE,* and Matt Staats, *Member, IEEE*

Abstract—Recommendation systems are intended to increase developer productivity by recommending files to edit. These systems mine association rules in software revision histories. However, mining coarse-grained rules using only edit histories produces recommendations with low accuracy, and can only produce recommendations after a developer edits a file. In this work, we explore the use of finer-grained association rules, based on the insight that view histories help characterize the contexts of files to edit. To leverage this additional context and fine-grained association rules, we have developed *MI*, a recommendation system extending *ROSE*, an existing edit-based recommendation system. We then conducted a comparative simulation of ROSE and MI using the interaction histories stored in the Eclipse Bugzilla system. The simulation demonstrates that MI predicts the files to edit with significantly higher recommendation accuracy than ROSE (about 63% over 35%), and makes recommendations earlier, often before developers begin editing. Our results clearly demonstrate the value of considering both views and edits in systems to recommend files to edit, and results in more accurate, earlier, and more flexible recommendations.

Index Terms— Programming environments/construction tools, interactive environments, software maintenance, data mining, association rules, programmer interaction histories

1 Introduction

Programmers spend a significant amount of time investigating files to edit. For example, Eclipse bug report #261613 shows that the programmer heavily investigated unrelated files for three days before editing just two files. The programmer wrote: "I think I'm getting closer to the real cause of the situation...." Similarly, in bug report #241244, programmers had a discussion over an investigation of "root causes" for two weeks, writing: "Further investigation still required…," and "… I'd like to investigate this direction further." These examples indicated that if programmers could find files to edit more easily, the time spent on software evolution tasks would be significantly reduced.

To assist programmers, researchers have developed history-based recommendation systems following two paradigms. The first group has mined software revision histories. Zimmermann et al. [38] and Ying et al. [36], for example, proposed recommending files to edit based on mined software revision histories. These approaches make file-to-edit recommendations by mining association rules between files frequently edited together in the past. The second group has mined programmer interaction histories. DeLine et al. [7] and Singer et al. [35], among others, proposed recommending the methods or files to view, based on mined programmer interaction histories. These approaches mine association rules between methods or files that past programmers viewed. These two paradigms have developed separately, leaving largely unanswered the question of which history is better to mine: view history or edit history.

This paper addresses this question. In this work, we evaluate *MI (M*ining programmer *I*nteraction histories), a recommendation approach considering both programmer edits and views. Our results shows that the records of files viewed by programmers help recommend files to edit. Using detailed view and edit histories to recommend files to edit produces the following benefits:

- **Accurate recommendations.** Viewed files provide more context when programmers edit, allowing more accurate recommendations over approaches which consider only edits.
- **Early recommendations.** Using view information allows recommendations to occur when programmers view files. Programmers can thus identify files to edit early, even before editing a single file.
- **Flexible recommendations.** When recommendations can occur based on viewed files, the recommendations change in response to programmers' navigation paths. This allows recommendations to occur even in scenarios that are not edit-heavy.

To empirically evaluate the benefits of considering views when recommending files, we evaluated MI against ROSE, an existing approach for recommending files to edit based on edit histories only [38]. MI extends ROSE, allowing us to produce recommendations from

- *Seonah Lee and Sungwon Kang are with the Department of Computer Science, KAIST, 373-1 Guseong-dong, Yuseong-gu, Daejeon, Republic of Korea. E-mail: {saleese, sungwon.kang}@kaist.ac.kr.*
- *Sung Kim is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China. E-mail: hunkim@cse.ust.hk.*
- *Matt Staats conducted this work at the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg. He is currently employed by Google, Inc. E-mail: staatsm@gmail.com*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

view and edit histories.

For our evaluation, we simulated code recommendations on 5,764 interaction traces stored in Eclipse Bugzilla [9]. The experimental results demonstrate that MI recommended files to edit with greater precision and recall than ROSE, with average precision and recall of 0.71 and 0.41 (respectively), compared to 0.41 and 0.28 for ROSE [38]. The results also demonstrate that MI makes recommendations in nearly half the time required for ROSE, with comparable levels of accuracy. We thus find that the use of view and edit histories—as represented by MI—is better both in terms of traditional metrics of recommendations accuracy and time to recommendation.

The remainder of this paper is organized as follows: Section 2 surveys related work. Section 3 proposes mining programmer interaction histories and the context formation that utilizes viewed files. Section 4 describes simulated evaluation. Section 5 discusses the results. Section 6 specifies threats to validity. Finally, Section 7 concludes the paper.

2 Related Work

A recommendation system for software engineering is "a software application that provides information items estimated to be valuable for a software engineering task in a given context" [32]. To recommend files relevant to conducting software evolution tasks, researchers have developed techniques mining the behavioral histories of programmers. The work can be classified into three areas: mining software revision histories, mining programmer interaction histories, and mining other data types.

Regarding the mining of software revision histories, researchers have used association rule mining, which finds rules among the occurrences of items in previous transactions [1]. To recommend files to edit, Zimmermann et al. and Ying et al. applied this technique to software revision histories [36][38]. Their approaches treat change sets as transactions and find association rules pertaining to the files frequently edited together in the past.

To mine semantically meaningful information, researchers have refined code edits. The first group introduced additional analysis. Fluri et al. extended the co-edited relationships of files using the structural relationships between the files [10]. Kim and Notkin proposed representing high-level changes with logic-based program analysis [19]. The second group introduced more fine-grained edits. Robbes et al. introduced the aspects of time, sequences and sessions [30]. Robbes et al. reconstructed task sessions by exploiting fine-grained edit information [29]. Later, Hattori et al. proposed Replay, which enabled programmers to observe fine-grained edits in the order of time [13][14]. Canfora et al. also suggested combining time information by introducing the Granger causality test [5]. The third group altered the mining target. Robillard and Dagenairs retrieved collections of code relevant to tasks by applying a nearest-neighbor clustering algorithm to software revision histories [33]. Kawrykow and Robillard proposed eliminating non-essential edits to extract only essential code edits [16]. Jaafar et al. mined a set of files edited together but not in the same change sets [15].

Regarding the mining of programmer interaction histories, researchers have also used association rule mining to recommend files to view. Parnin and Görg mined association rules in programmer interaction histories [26]. Likewise, DeLine et al. proposed TeamTracks [7] and Singer et al. proposed NavTracks [35], which use associations between files that past programmers viewed for predicting the next files to view. These approaches showed lower recommendation accuracy than the approaches mining software revision histories. Meanwhile, Kersten and Murphy proposed Mylyn [17] for recommending files relevant to programmers' tasks, based on the frequencies of interactions that a programmer has with files. A shortcoming of Mylyn is that it is a semi-automated approach in that it relies on programmers' manual identification and indication of tasks; users who lack prior knowledge about tasks and their contents are unable to determine which collection of code is relevant to their current task.

The recent work in this area can be divided into two groups. The first focuses on improving recommendation accuracy. To evaluate the accuracy of code recommendations, Robbes et al. proposed replaying programmer interaction histories with a revised cumulative gain [31]. Piorkowski et al. studied several recommendation models. They collected the interaction traces recorded while undergraduate students performed two tasks on different code bases within two hours [27]. By repeating predefined tasks, they measured the recommendation accuracy of different models. Piorkowski et al. also proposed the PFIS (Programmer Flow by Information Scent) recommendation model based on information foraging theory [28], and compared the PFIS models to TF/IDF based recommendation models. The second uses the same interaction data we use in this paper. Ying and Robillard analyzed programmer interaction histories and revealed the relationships between task types and edit patterns [37]. Lee et al. extracted 56 metrics from programmer interaction histories and created a classification model to predict files that include defects [23]. Lee and Kang compared their approach with TeamTracks, using the interaction histories [21][22].

Researchers have diversified the types of recommendations and mining data. For example, Bacchelli et al. proposed mining e-mails to recommend e-mails related to given program elements [3]. Sawadsky et al. proposed mining web pages to recommend the web pages to be revisited, related to the code [34]. Kim et al. proposed mining bug reports to recommend files to fix related to a given bug report [18]. It is noted that Kim et al.'s work is closely related to our work in that their approach recommends files to fix prior to fixing bugs. However, the approach still yields low recommendation accuracy, ranging from 7~11% precision values[1], and the target mining data are different from ours.

[1] This means that, when it recommends ten files to fix, one of them is correct and the other nine are incorrect.

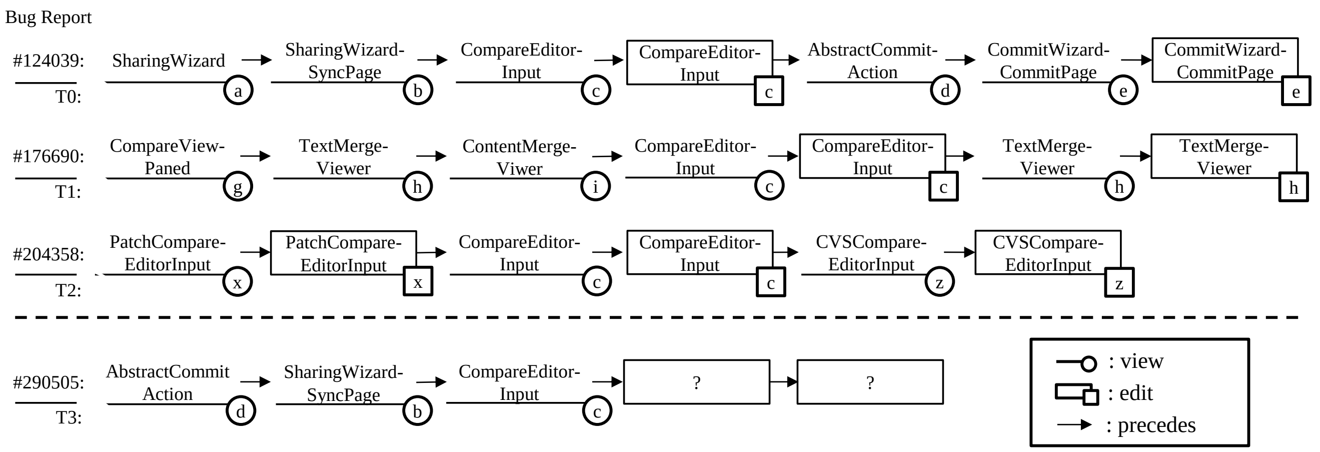


Fig. 1. An example that shows the files programmers view and edit while performing tasks. This example is simplified from the actual interaction traces of bug reports #124039, #176690, #204358, and #290505 in the Eclipse Bugzilla system [9].

Our work differs from previous work in that ours illustrates that mining the records of viewed files, along with edited files, can significantly improve recommendation performance. Furthermore, our work differs from Kim et al. [18] in that our research emphasis is on the viewed files of programmer interaction histories, while theirs is on bug reports, which are different data sets. Our evaluation is focused on revealing the benefits of viewed histories, and our results show much higher recommendation accuracy than theirs (about 63% over 11%).

3 Mining Programmer Interaction Histories

Programmer interaction histories contain the records of files viewed by programmers as well as those edited, and, are thus a more informative source than software revision histories. We believe this can significantly improve recommendations for editing, as the records of viewed files establish a more accurate programmer context to recommend relevant files to edit.

Let us explain this idea with the example in Fig. 1, which shows a situation in which programmers have performed tasks T0, T1, and T2, and a programmer is now performing task T3. The lower-case letters in Fig. 1 represent the files that are viewed or edited by programmers. While performing T3, this programmer views files *d*, *b*, and *c*. Given these interaction events, the following question arises: what are the files that the programmer is likely to edit for T3?

Previous approaches that mine edit histories (e.g., ROSE [38]) cannot make recommendations at this point because the programmer has not yet edited a file. If the programmer edits a file, the files {c} for example, ROSE utilizes {c} as a context to find other files to edit together. As {c} was edited in T0, T1 and T2, ROSE will recommend all other files edited in T0, T1, and T2, {e, h, x, z}.

In contrast, the approach for mining programmer interaction histories that we propose, *MI* (*M*ining programmer *I*nteraction histories), recommends files to edit using the context provided by both the viewed files and the edited files. The records of viewed files are used to identify the interaction history events that are most similar to T3. As the programmer's current navigation activities for T3 are most similar to T0, the files edited in T0, i.e. {c, e}, are most likely to be edited in T3. When the programmer views *d*, *b* and *c*, MI uses the viewed files {d, b, c} to find files to edit. As the files {d, b, c} were also viewed in T0, MI recommends the files that were edited in T0, {c, e}. Once a programmer edits {c}, MI adds this interaction to create the context of viewed files {d, b, c} and edited file {c}. MI again associates this context with T0 and recommends file {e} to edit.

Thus, we hypothesize the following:

Using a context that includes viewed files can improve the accuracy of recommending files to edit.

To recommend files to edit by utilizing the records of viewed files, MI mines interaction histories. As shown in Fig. 2, MI mines interaction traces, finds association rules using the current context, and generates recommendations of files to edit. The essential part of the recommendation system is the *context*. The context characterizes the situation of the programmer (e.g., viewed files), and is used as a query at the time of recommendation [8][32]. We first explain the detailed procedure of MI in Section 3.1 and propose the outline for context formation in MI in Section 3.2.

3.1 MI

MI extends ROSE [38]. The original ROSE is an approach which mines software revision histories [38]. We have revised ROSE to mine programmer interaction histories. This revised ROSE mines the association rules from edited files in programmer interaction histories and forms a context using only edited files. By extending this version of ROSE to include viewed files, we propose mining association rules in programmer interaction histories to recommend files to edit (MI). MI mines the association rules

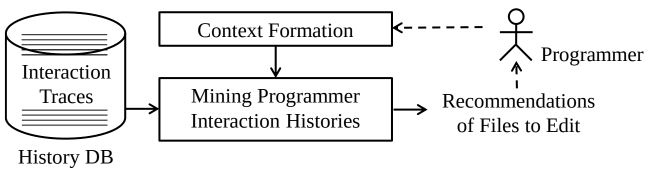

Interaction Traces
History DB
Context Formation
Mining Programmer Interaction Histories
Programmer
Recommendations of Files to Edit

Fig. 2. Overview of the proposed recommendation system MI

from viewed and edited files, forming a hybrid context consisting of viewed and edited files. Our MI approach can use several methods to form a context from viewed and edited files, each explained in Section 3.2. We empirically evaluate the effectiveness of these various methods in Section 4.

3.1.1 Interaction Traces

An interaction trace is a log consisting of records that describe a programmer's actions (i.e. views and edits) and files on which the actions were taken. An interaction trace can be expressed as T_k, where k represents a software evolution task that a programmer performed. An interaction trace T_k is converted into a pair of sets: $T_k = (V_k, E_k)$, where V_k is the set of viewed files in T_k, $V_k = \{v_1, ..., v_n\}$ and E_k is the set of edited files in T_k, $E_k = \{e_1, ..., e_m\}$. The collections of interaction traces can be expressed as *History DB* = $\{T_k \mid 1 \leq k \leq i-1\}$.

3.1.2 Context

Conceptually, a context is "any information which can be used to characterize the situation of" a current user [8]. In a recommendation system, a context becomes a query, which triggers a recommendation [32].

In MI, a context is formed from a current programmer's actions. When the current programmer is performing a task i, a context is created from the last files viewed and edited by the current programmer from each timepoint in T_i. As the current programmer continues viewing and editing files, the context changes. The context C can be expressed as (V_c, E_c), where V_c is a set of the last v files that a current programmer has viewed, $V_c = \{v_1, ..., v_v\}$, and E_c is a set of the last e files that the programmer has edited $E_c = \{e_1, ..., e_e\}$ at each timepoint. Section 3.2 will explain the details of forming a context.

3.1.3 Mining on the Fly

Given a context C, the interaction traces collected in History DB can be mined on the fly [38]. As context C narrows the scope to mine, this on-the-fly mining approach is a reasonably lightweight process.

MI mines the association rules[2] in the form of $(V_c, E_c) \Rightarrow \{e\}$ where $e \not\in E_c$ and $e \not\in V_c$. The antecedent (V_c, E_c) must have occurred together at least in one interaction trace in History DB. The consequent $\{e\}$ is one of the other edited files in the interaction traces that contain the antecedent (V_c, E_c) in History DB. Because the antecedent and the consequent of an association rule are disjoint, the files that belong to $C = (V_c, E_c)$ are excluded from the consequent.

MI first sets up the antecedent as context $C = (V_c, E_c)$ before mining begins. MI then finds association rules by checking whether each interaction trace T_k contains both of the viewed files V_c and edited files E_c of context C: $V_c \subseteq V_k$ and $E_c \subseteq E_k$. Subsequently, MI counts all edited files in the interaction traces satisfying the condition as the consequent {e}, where $e \not\in E_c$ and $e \not\in V_c$. Finally, MI returns the list of consequents.

[2] An association rule consists of an antecedent X and a consequent Y. The antecedent X is the condition in which the association rule can be applied, and the consequent Y is the result that the association rule can produce. An association rule is defined as $X \Rightarrow Y$, where X and Y belong to DB and are disjoint [1].

3.1.4 Ranking and Recommendation

To rank the association rules found, MI uses the concepts of support and confidence, as typically used in association rule mining [1][12]. Support refers to the number of co-occurrences of the antecedent and the consequent of an association rule in History DB. Confidence is the ratio of the co-occurrences of the antecedent and the consequent to the occurrences of the antecedent.

To calculate the support, MI counts the number of interaction traces that include both the antecedent (V_c, E_c) and the consequent $\{e\}$. To calculate the confidence, MI divides the support by the number of interaction traces including the antecedent (V_c, E_c). MI has minimum support and minimum confidence thresholds and selects association rules that meet these thresholds. MI ranks the consequences by confidence. According to this ranking, MI finally recommends the files to edit.

3.2 Context Formation

For recommendation systems, context is defined as "the information about the user, their environment and their work that are available at the time of recommendation" [32]. We believe that a key challenge in recommendation systems is to determine what user contexts are best suited to producing recommendations useful to users.

To create a context C in Section 3.1.2, MI uses a sliding window, referred to here as a *v-e*-sized sliding window. The sliding window consists of two queues that hold v different viewed files and e different edited files. The sliding window disregards reappearing records of the same action on the same file as held in the sliding window at each time. Whenever the sliding window is updated with a new file in either queue, it creates a context that consists of the last v viewed files and e edited files. Fig. 3 illustrates how a context C is formed from an interaction trace T_i. For example, the *3-0*-sized window traces up to three different viewed files and zero edited files. When the sliding window initially captures three viewed files {a, b, c}, it creates the context ({a, b, c}, ∅). When the programmer views file {d}, the sliding window obtains a new viewed file {d} and creates the context ({b, c, d}, ∅).

For systems recommending files to edit, several con-

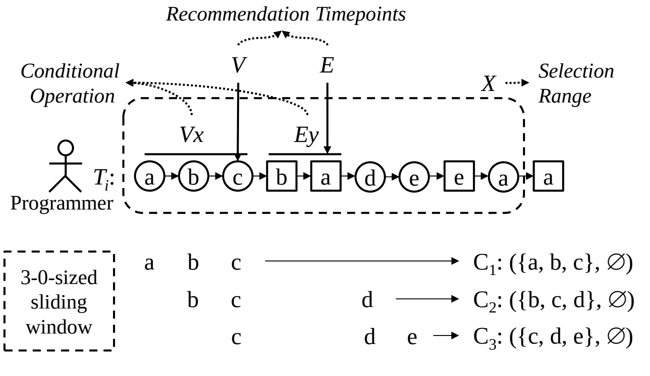


Fig. 3. Forming a context

texts are possible, and relate to the choice of information and how selected information should be combined. This section discusses how contexts can be formed as follows: Section 3.2.1 outlines the rules of combining viewed files and the edited files; Section 3.2.2 proposes four different methods of forming a context; and Section 3.2.3 explains how each method can be used for improving recommendation performance.

3.2.1 Rules for Forming a Context

We establish three rules to combine different kinds of information, e.g., viewed files Vx = {a, b, c} and edited files Ey = {b, a}, as shown in Fig. 3.

Conditional operation rule. If a context is formed from different kinds of information (e.g., user actions), a conditional operation is needed to combine them into a context. For example, if a programmer views files Vx and edits files Ey, Vx and Ey can be combined with the *OR* or *AND* operation, denoted respectively as:

- AND (Vx, Ey): this context makes a recommendation when it finds both Vx and Ey in the mined rules.
- OR (Vx, Ey): this context makes a recommendation when it finds either Vx or Ey in the mined rules.

Selection range rule. When a context is formed, a range can be given to increase the chance of creating recommendations at timepoints. The range starts from the timepoint for characterizing a situation and ends at the timepoint for a recommendation. When the range is set to x and the number of viewed files in a context C (i.e. V_c) is set to v, a context can be formed by selecting v files from the last x files that a current programmer has viewed (i.e. ${}_xC_v$). For example, if x is set to 5 and v is set to 3, the context finds any combination of three viewed files from a programmer's most recent five viewed records in the mined rules. The relevant rules can be easily found by using a set operation, $|Vx \cap V_k| \geq v$. In Fig. 3, when a programmer views {e}, this range setting makes a difference. While the 3-0-sized sliding window makes a recommendation only with a context {c, d, e}, the sliding window with the range setting makes a recommendation with ${}_5C_3$ (= 10) cases from {a, b, c} to {c, d, e}. Because programmers usually do not visit and edit files in the same order, this increases the chance to create a recommendation at each timepoint. A range is denoted as:

- RANGE (X): when a range has the fixed number X, a programmer's interactions are traced up to X records.
- RANGE (ϕ): when a range is not set up, only the last interactions to be used as a context are traced up.

Recommendation timepoint rule. When different kinds of user actions are monitored, certain kind of actions can be selected as the triggers for recommendations. For example, if a programmer views and edits files, recommendations can be created at view points, edit points, or view and edit points, denoted respectively as:

- POINT (V): a recommendation is triggered at each view point
- POINT (E): a recommendation is triggered at each edit point
- POINT (V, E): a recommendation is triggered at each view and edit point.

TABLE 1
FOUR METHODS OF FORMING A CONTEXT

	AND (Vx, Ey)	OR (Vx, Ey)
POINT (E)	MI-EA-RANGE (X)	MI-EO
POINT (V, E)	MI-VA	MI-VO

In our approach, these proposed rules are the basis of forming contexts. The most important rule is the conditional operation, because it directly addresses the items of a context. Selection range is subsidiary because it allows the conditional operation to randomly select items from a range. The recommendation timepoint is important because it determines the time of recommendation.

3.2.2 Methods for Forming a Context

A combination of conditional operation rules and recommendation timepoint rules can create the four possible methods for forming a context with viewed and edited files, as shown in Table 1.

- MI-EA-RANGE(X): this forms a context by combining viewed and edited files with the *AND* operation, and creates a recommendation at each edit point. When edit points trigger recommendations, the range option for viewed files should be used to increase the chance of the recommendations.
- MI-EO: this forms a context by combining viewed and edited files with the *OR* operation, and creates a recommendation at each edit point. In this case, the range rule is not needed, because the *OR* condition will create a recommendation with only the edit files, and the number of edited files is typically limited.
- MI-VA: this forms a context by combining viewed and edited files with the *AND* operation, and creates a recommendation at each view or edit point. Note that because the method creates a recommendation when obtaining both viewed and edited files, this method creates a recommendation after an edit point.
- MI-VO: this forms a context by combining viewed and edited files with the *OR* operation, and creates a recommendation at each view or edit point. Because the method can create a recommendation when obtaining viewed or edited files, this method can create a recommendation even before an edit point.

3.2.3 Relationships between Methods Forming a Context and Recommendation Results

The context used directly impacts the effectiveness of recommendation results. For example, the *AND* conditional operation may yields higher recommendation accuracy than the *OR* operation, because the *AND* operation selects the mined rules more narrowly than the *OR* operation. Likewise, the use of edit or view points affects the frequency of recommendations because it determines whether a recommendation occurs only at view points, edit points, or both view and edit points.

With this in mind, we discuss which context is likely to be more effective, in the sense that using it leads to more accurate, earlier and more flexible edit recommendations.

For accurate recommendations, we use MI-EA-RANGE (X) or MI-VA, because both use the *AND* operation and thus mines rules narrowly. The difference is that MI-EARANGE(X)

TABLE 2
INTERACTION TRACES OF 72 PROJECTS

Project	Description	Period	#Interaction Traces	File Level		
				#Views/ #Traces	#Edits/ #Traces	View-edit ratio
Mylyn	A task management tool for programmers	2006-05-18~2011-07-07	2,726	14.4	2.8	5.1
Platform	Eclipse core frameworks	2007-10-18~2011-05-25	582	24.2	2.7	9.0
PDE	Plug-in development environment	2007-11-11~2011-02-25	536	6.4	1.1	5.8
ECF	Eclipse communication frameworks	2007-04-06~2011-06-18	308	9.3	0.7	13.3
MDT	Modeling development tools	2010-01-26~2011-05-20	245	54.5	0.7	77.9
Others (67)	Other projects (i.e. Equinox, GEF)	2008-06-13~2011-06-21	1,367	22.8	1.4	16.3
Total	All of the above projects (72)	2006-05-18~2011-07-07	5,764	18.1	2.1	8.6

These interaction traces are from the Eclipse Bugzilla system [9].

creates recommendations only at edit points, whereas MI-VA creates recommendations at view and edit points.

For early recommendations, we use MI-VO, because it is the only method that creates recommendations at each view point before an edit. For flexible recommendations, we can use both MI-VO and MI-VA, because this allows us to create recommendations at view and edit points.

We do not explore the use the MI-EO method, because MI-EO creates recommendations at edit points using the *OR* operation. This does not leverage the benefits of considering view context, and thus is very unlikely to contribute to accurate, early, or flexible recommendations.

Then the remaining problem is that the above methods likely do not produce both early and accurate recommendations. For example, MI-VO, which is used for early recommendations, is unlikely to yield particularly high recommendation accuracy because it uses the *OR* operation. In contrast, MI-VA, which is used for accurate recommendations, makes recommendations after the first edit, because of the *AND* operation. We thus combine these two methods and call the combined method as MI-VOA.

MI-VOA uses MI-VO and MI-VA in the following way:

```
if (both Vx and Ey exist) then
        apply MI-VA // make recommendations after an edit
else
        apply MI-VO // make recommendations before an edit
end if
```

4 SIMULATION EVALUATION

We performed an experiment to evaluate the effect of using viewed files on the recommendation results. Section 4.1 explains the experiment design, and Section 4.2 presents the experimental results.

4.1 Experiment Design

We designed a simulated comparative controlled experiment [24]. Section 4.1.1 identifies the research questions, Section 4.1.2 identifies the variables, Section 4.1.3 identifies the subjects to be used, Section 4.1.4 describes the detailed procedure, and Section 4.1.5 presents the metrics for the evaluation.

4.1.1 Research Questions

We are interested in studying the effect of mining the records of viewed files, and showing utilizing the increased amount of information in the records of viewed files improves edit recommendations. We explored four research questions:

RQ1. Can using viewed files improve the recommendation accuracy?

RQ2 Can using viewed files (MI) create more accurate recommendations than using only edited files (ROSE)?

RQ3 Can using viewed files (MI) create earlier recommendations than using only edited files (ROSE)?

RQ4 Can using viewed files (MI) create more flexible recommendations than using only edited files (ROSE)?

To evaluate these questions, we use variations of MI designed to be "fair" with respect to ROSE. For example, when comparing accuracy, we use MI-EA, which makes recommendations only at edit points; otherwise, we measure accuracy at very different timepoints. The final question is based upon our final recommendation, MI-VOA.

4.1.2 Independent and Dependent Variables

In each research question, the independent variable is a context the approach used, while the dependent variables are measurements related to the quality of the recommendations results.

For the independent variable, we explore five different approaches: four variations of MI, and ROSE. These approaches vary based on the context *C*, which we outline in Table 3. Specifically, *C* is defined based on three components:

- The methods for forming a context, especially the one which includes viewed files or the other
- The number of viewed files *v*
- The number of edited files *e*.

We detail how we control the context *C* in Sections 4.1.4.2 through 4.1.4.5.

Our dependent variables are measurements of recommendation effectiveness, e.g., accuracy of the recommendations, speed at which recommendations are made, etc. Specifically, we measure:

- *Precision*. Higher precision indicates more recommendations are correct, i.e. should in fact be edited.
- *Recall*. Higher recall indicates that more files that should be edited have been recommended.
- *F-measure*. F-measure combines precision and recall,

TABLE 3

FRAMEWORK TO COMPARE FORMING CONTEXTS WITH EDITED FILES AND VIEWED FILES

		ROSE (e-edits)	MI (v-views, e-edits)			
			RQ1: MI-VA	RQ2: MI-EA	RQ3: MI-VO	RQ4: MI-VOA
Context C		$C = (\varnothing, E_c)$ where E_c is the set of edited files $\{e_1, \ldots, e_e\}$	$C = (V_c, E_c)$ where V_c is the set of viewed files $\{v_1, \ldots, v_v\}$ and E_c is the set of edited files $\{e_1, \ldots, e_e\}$			
Context Formation Method	*Conditional Operation Rule*	*NA*	AND (*Vx*, *Ey*)	AND (*Vx*, *Ey*)	OR (*Vx*, *Ey*)	*if (Vx and Ey exist) then MI-VA else MI-VO end if*
	Selection Range Rule	RANGE($\varnothing$)	RANGE($\varnothing$)	RANGE(*100*)	RANGE($\varnothing$)	RANGE($\varnothing$)
	Timepoint Rule	POINT(*E*)	POINT(*V*, *E*)	POINT(*E*)	POINT(*V*, *E*)	POINT(*V*, *E*)
Rationale		*A baseline*	*A basic method with view histories*	*Recommendations at edit points*	*Recommendations even only with viewed events.*	*Better work than ROSE across all RQs.*
Measures		*Precision Recall*	*Precision Recall ROC*	*Precision Recall F-Measure*	*#of interactions before recommendation*	*#of recommendations within the first 100 interactions*
Finding		*This ROSE is comparable to the original ROSE*	*MI-VA yields higher accuracy than ROSE*	*MI-EA yields higher accuracy than ROSE*	*MI-VO yields earlier Rs than ROSE*	*MI-VOA works better than ROSE across all RQs*

The number of viewed files that C includes is identified as v and the number of edited files is denoted as e. NA is "not applicable", R is "recommendation," and RQ is "Research Question."

and is used to measure the general accuracy of the recommendations.

- *Feedback.* Higher feedback indicates the recommender system is more efficient with respect to the number of queries taken.
- *Receiver Operating Characteristic.* Illustrates how well the method finds true positives and true negative recommendations.
- *#of interactions before recommendation.* To determine how quickly the system produces recommendations, we have computed the average number of interactions before a recommendation is made.
- *#of recommendations within the first 100 interactions.* To determine how frequently the system produces recommendations, we have computed the average number of recommendations within certain number of interactions.

We detail how each measurement was made in Section 4.1.5.

4.1.3 Subjects

We used Mylyn data [17]. Mylyn records interaction traces from the Eclipse IDE and archives them in the Eclipse Bugzilla system [9]. We extracted 5,764 Mylyn interaction traces from 72 Eclipse sub-projects. Table 2 presents those projects including the five major projects: Mylyn, Platform, PDE, ECF, and MDT. The 67 other projects have an average of 20 interaction traces.

We identified viewed files and edited files from the Mylyn interaction traces. Mylyn records interaction events in XML format, each of which consists of several fields (e.g., <InteractionEvent StartDate="2009-01-24 10:40:28.321 MST" EndDate="2009-01-24 10:40:28.321 MST" StructureHandle="…Property.java" Kind= "selection"… />). There are types of events, such as selection and edit [17]. To identify viewed files, we used the records that have the *selection* event type. The selection-type events occur when programmers click on files in the Eclipse IDE. To identify edited files, we used the records that have the *edit* event type. The edit-type events, however, occur not only when a programmer actually edits files but also when double-clicking on a file to open it in a code editor. Therefore, the edit type events do not necessarily indicate actual edits [22]. Fortunately, when Mylyn records programmer's double clicking as an edit event, its starting time and ending time are the same [22]. We thus identified the files that contain only a record of edit events whose starting and ending times are the same as viewed files, and the files that contain a record of edit events whose starting and ending times are different as edited files.

The MI recommendations exploit the fact that programmers usually view many files other than the files to edit. To check this, we counted the numbers of viewed files and edited files, as shown in the three rightmost columns of Table 2. The average number of viewed files is about 18, whereas the number of edited files is about 2. The average ratio of viewed files to edited files is about 8.6. This ratio indicates that programmers view seven to nine files while editing one file.

4.1.4 Evaluation Methodology

For our evaluation, we chose a simulation-based comparative controlled method [24]. This method allows us to maintain the same conditions for a fair comparison of MI and ROSE. Table 3 presents a comparison framework, where MI includes viewed files but ROSE does not. We simulated these approaches to recommend the files to edit with the Mylyn data described in Section 4.1.3.

4.1.4.1 Set-up for Simulation

To simulate recommendations, we need a training set, a test set and a simulator. A training set is the interaction traces from which association rules are mined. A test set is the interaction traces, each of which provides the answer set of the programmer's actual edited files. To identify a training set and a test set, we employed an online machine-learning approach [20], in which every interaction trace belonging to a subject is first used as a test set and next used as a training set. The initial interaction trace T_1, which has no training set, does not need to be a test set. Each interaction trace from T_2 to T_n, where n is the number of interaction traces belonging to a subject, is used as a test set once before becoming a training set. When an interaction trace T_i is used as a test set, the interaction traces which occurred prior to T_i, from T_1 to T_{i-1}, are used as the training set, as in earlier work [38].

The simulator creates multiple contexts from test set T_i. To create a context C, the simulator moves a *v*-*e*-sized sliding window from the first to the last record of T_i (Refer to Section 3.2). By setting the numbers of *v* and *e* files, the sliding window can hold *v* viewed files, *e* edited files, or both. Once the context C is created from test set T_i, the simulator mines association rules through a training set from T_1 to T_{i-1} (Refer to Section 3.1.3). Based upon mined rules, the simulator recommends files to edit. To select and rank mined rules, we used the same options used in Zimmermann et al. [38]: the minimum support is 1 and the minimum confidence is 0.1. If a recommendation includes more than 10 files to edit, the simulator only recommends the 10 top-ranked files.

We now outline the processes used to evaluate each research question. Note that when evaluating research questions 2 through 4, we set the number of viewed files to three (v = 3) and the number of edited files to one (e = 1). Considering that programmers view an average of 7~9 files to every one file edited, as Table 2 shows, using more viewed files than edited files is reasonable.

4.1.4.2 Experiment for RQ1

To investigate the effect of viewed files on recommendation accuracy in general, we explored the impact of the context *C*:

- $C = (\varnothing, E_c)$ *for control group*: this forms a context using only edited files. This is ROSE (*e*-edits), and used as the baseline.
- $C = (V_c, \varnothing)$ *for comparison group*: this forms a context consisting of only viewed files. This (*v*-views) is used to identify the characteristics of using viewed files as a context.
- $C = (V_c, E_c)$ *for treatment group*: This forms a hybrid context consisting of viewed and edited files. This is used to show the effect of viewed files as additional information for a context. This is MI (*v*-views and *e*-edits), combining *v*-views and *e*-edits with the AND operation (i.e. MI-VA).

With these alternative methods, we observed the performance changes, as measured by precision and recall, when varying *v* and *e* values. We set $v = e = n$, increasing *n* from 1 to 10, i.e. increasing *v* from 1 to 10 for "*v*-views", *e* from 1 to 10 for "*e*-edits," and both *v* and *e* from 1 to 10 for "*v*-views and *e*-edits." For each value of *n,* our simulation ran over all of the 5,674 interaction traces repeatedly. Furthermore, we also examined Receiver Operator Characteristic to visualize the impact of varying *v* and *e.*

4.1.4.3 Experiment for RQ2

To investigate the impact of viewed files on the accuracy of recommended files, we again compared ROSE (context $C = (\varnothing, E_c)$) against MI-EA-$_{\text{RANGE(100)}}$. Recall from Section 3.1.3 that MI-EA, like ROSE, makes recommendations only at edit points [38]. This allows for a fair comparison of accuracy, as we compare only simultaneously made recommendations.

As noted previously, we set the window for the number of viewed files (*v*) to three in our studies. However, if MI-EA used only the three viewed files prior to an edit file as a context, few recommendations would occur. For example, when the current programmer views {c, d, e} and edits {e}, a recommendation will occur only in the case previous programmers viewed views {c, d, e} and edits {e}. This restricts the occurrences of recommendations too much at each edit point. To increase the chance to create a recommendation at each edit point, we used the selection range rule. We used MI-EA-$_{\text{RANGE (100)}}$, which MI-EA tracks up to one hundred records of viewed files prior to an edit point, and forms a context of three viewed files from the tracked records with an edited file.

We evaluate the accuracy of recommended files using precision, recall and F-measure.

4.1.4.4 Experiment for RQ3

To investigate the impact of viewed files on the timing of recommendations, we compared ROSE with MI-VO, which can make recommendations on views or edits. This allows the possibility of earlier recommendations than MI-EA (used in RQ2) and potentially ROSE.

In evaluating this research question, we used the transactions that include both viewed and edited files as a test set. We then examined the first recommendation occurring in each transaction. We counted the number of interaction events occurring to the first recommendation. We averaged the counts for all of the recommendations.

The resulting averages were then compared for ROSE and MI-VO to determine which approach produces earlier recommendations.

4.1.4.5 Experiment for RQ4

To investigate the impact of viewed files on the flexibility of the recommendations, we compared ROSE with MI-VOA. MI-VOA makes recommendations using both OR and AND-based recommendation rules, and thus is intended to combine the benefits of both approaches.

To measure the flexibility of recommendations, we required a metric that blended the timing of the recommendations with the precision and recall of said recommendations. For this, we checked m-th recommendations and averaged their recommendation points and their F-measure values. We then examined the averaged m-th recommendations occurring at the recommendation points prior to 100th interaction event.

4.1.5 Measurement

To measure the recommendation accuracy, we used several commonly used metrics. First, *precision* and *recall* are widely used to evaluate the effectiveness of information retrieval approaches [12]. To compute them, we used the recommended files (A) and the files actually edited (E).

- A: Set of the files recommended to edit which are driven by context C in interaction trace T_i. Because we only recommend up to the 10 top-ranked files, $|A|$ is always less than 10.
- E: Set of the files that are actually edited in interaction trace T_i except the edited and viewed files of context C. $|E|$ varies with T_i, because a different T_i will have a different number of edited files.

The metrics precision P and recall R are calculated for a recommendation by using the formulas as below. If there is more than one recommendation, the precision and recall values for a recommendation are calculated first and then averaged across all recommendations, which is called the *macro evaluation* technique [38]:

$$Precision\ P = \frac{|A \cap E|}{|A|} \qquad Recall\ R = \frac{|A \cap E|}{|E|}$$

As there are trade-offs between precision and recall, it is difficult to compare the accuracy using just precision and recall values. *F-measure* is the harmonic mean of precision and recall, and allows us to measure recommendation accuracy while capturing this tradeoff. F-measure is computed from the averaged values of precision and recall as follows:

$$F\text{-}measure\ F = \frac{2 * P * R}{P + R}$$

To visualize the how the accuracy is impacted by varying v and e, we used the Receiver Operating Characteristic (ROC), and calculated the Area Under Curve (AUC) [10]. To compute them, we used the sensitivity and specificity changes by varying v and e values.

- Sensitivity: This shows how well a method finds the true positives. It is calculated by dividing the number of true recommendations yielded over each v and e setting by the total number of true recommendations for all settings.
- Specificity: This shows how well a method finds the true negatives. It is calculated by dividing the number of false recommendations yielded under each v and e setting by the total number of false recommendations for all settings.

ROC is a curve plotting 1-Specitivity on x-axis and Sensitivity on y-axis. AUC[3] is calculated by summing up the sensitivity values when specificity values change in the ROC curve, and by dividing the sum by the number of the specificity values changing in the curve [10].

$$AUC = \frac{\sum Sensitivity\_when\_Changing\_Specificity}{\#Points\_of\_Changing\_Specificity}$$

To measure the general efficiency of the recommendation systems, we used *feedback*, as used by in Zimmermann et al. [38]. Feedback measures the percentage of queries which yield recommendations. Feedback Fb is the number of overall recommendations divided by the number of queries:

$$Feedback\ Fb = \frac{\#Recommendations}{\#Queries}$$

To determine if MI creates earlier recommendations than ROSE, we check the timepoints of creating the first recommendations of MI and ROSE. We counted how many interaction events occurring to the first recommendation, and averaged the counts.

$$Nth = \frac{\sum \#Interaction\ \_Events\_to\_Recommendations}{\#Recommendations}$$

To determine if MI creates more frequent recommendations than ROSE, we first compare the number of recommendations of MI and ROSE, and then investigate the recommendations occurring in early interaction steps up to 100 interactions.

4.2 Results

This section presents the results of our evaluation. We reserve a larger discussion of the implications of our results for Section 5.

4.2.1 RQ1: Can using viewed files improve the recommendation accuracy?

To investigate the recommendation performance of mining viewed and edit histories, we observed the performance changes using various v and e values by setting $v = e = n$ and increasing n from 1 to 10.

4.2.1.1 Precision and Recall Graph

Fig. 4 presents the resulting precision and recall graph. Curves "n-views," "n-edits," and "n-views and n-edits" represent the recommendation results of using viewed files ($v = n$), edited files ($e = n$) and both files ($v = n$ and $e = n$) as a context. The first point of each curve closest to the bottom represents the result when n is 1. For example, the first point of the "n-views" represents the results of using 1 viewed files as a context ($v = 1$), whose averaged precision is 0.2 and recall is 0.3. The first point of the "n-edits" represents the results of ($e = 1$), the precision is 0.4 and recall is 0.3. The first point of the represents the results of ($v = 1$ and $e = 1$), the precision is 0.5 and recall is 0.4. The points of each curve go up as n increases.

Fig. 4 shows that "n-views and n-edits" achieves consistently higher precision than "n-views" or "n-edits" in the y-axis, while "n-views" shows consistently lower precision than "n-edits." For example, "1-view and 1-edit" yields higher precision than "1-edit" (0.5 vs. 0.4), and "1-edit" yields higher precision than "1-view" (0.4 vs. 0.2). "2-views and 2-edits" yields higher precision than "2-edits" (0.8 vs. 0.7), and "2-edits" yields higher precision than "2-views" (0.7 vs. 0.3).

[3] As ROC and AUC measures the accuracy of classification models, these do not perfectly fit to our recommendation models. However, as AUC shows a simple value for representing its accuracy, we tried to adjust ROC and AUC in order to measure recommendation accuracy.

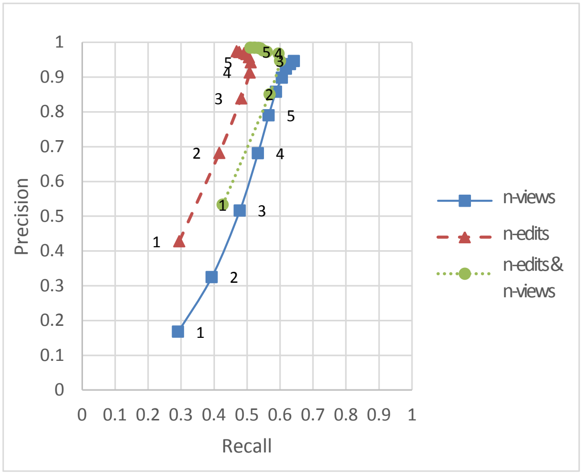


Fig. 4. Precision and recall graph

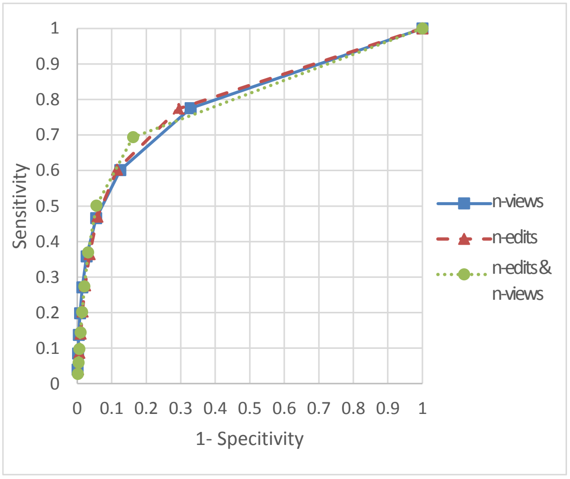


Fig. 5. Receiver Operating Characteristic Curve

Fig. 4 also shows that "*n*-views and *n*-edits" yields consistently higher recall than "n-edits," whereas, when n is larger than 5, "*n*-views" yields higher recall than the others. For example, "*1*-view and *1*-edit" yields higher recall than "*1*-edit" (0.4 vs. 0.3) and "*1*-view" (0.4 vs. 0.3). "2-views and 2-edits" yields higher recall than "2-edit" (0.6 vs. 0.4) and "2-view" (0.6 vs. 0.4). "3-views and 3-edits" reaches the peak in the curve (0.6). However, while the recall of "*n*-views" continues to increase as n increases, the recall of "*n*-views and *n*-edits" and "*n*-views" is typically comparable, within 0.1.

As "*n*-views and *n*-edits" yields consistently higher precision and recall than "*n*-edits," we conclude:

Using viewed and edited files ("n-views and n-edits") yields consistently higher file-level recommendation accuracy than using only edited files ("n-edits").

<u>4.2.1.2 Receiver Operating Characteristic Curve & AUC</u>

Fig. 5 presents the results for RQ1 in ROC. In Fig. 5, the three curves are very similar to each other. Thus we can see that whenever the number of views, edits, or both is increased, the resulting improvement in recommendation accuracy is about the same.

This may seem to contradict Fig. 4, because Fig. 4 shows a rapid increase for "*n*-view" in precision. However, these results occur only because the ROC results reflect largely the increment rate of the effect corresponding to a particular change. Fig. 4 shows that the improvement of recommendation accuracy becomes lower whenever the number of views, edits, or both is increased. We note that all of the three ROC curves show this trend. We also note that each ROC curve has a different baseline for recommendation accuracy (0.2 precision for "*n*-views," 0.4 for "*n*-edits," and 0.5 for "*n*-views and n-edits"), which might be related to that the curve of "*n*-views and *n*-edits" is similar to those of "*n*-views" and "*n*-edits."

The AUC[4] value of "*n*-views" is 0.58, that of "*n*-edits" is 0.58, and that of "*n*-views and *n*-edits" is 0.52. Given these comparable values, we conclude:

Using additional viewed files ("n-views") achieves the same improvement of recommendation accuracy as using additional edited files ("n-edits").

4.2.2 ROSE Recommendation Results

In our experiment, ROSE yields 24,768 queries and 14,924 recommendations. Table 4 shows the results based on precision, recall and F-measure. For example, in the Platform project, ROSE recommended files to edit with 0.58 precision (P), 0.25 recall (R), and 0.35 F-measure (F). On average, ROSE yielded 0.41 precision (P), 0.28 recall (R),

TABLE 4
ROSE RESULTS OF MINING INTERACTION HISTORIES

	ROSE ($e = 1$)					
Project	P	R	F	Fb	#Rs	#Qs
Mylyn	0.34	0.27	**0.30**	0.70	10,753	**15,248**
Platform	0.58	0.25	**0.35**	0.56	1,974	**3,543**
PDE	0.36	0.32	**0.34**	0.32	321	**988**
ECF	0.25	0.25	**0.30**	0.34	119	**352**
MDT	0.90	0.37	**0.52**	0.35	116	**336**
Others	0.60	0.30	**0.42**	0.38	1,641	**4,319**
Average	0.41	0.28	**0.33**	0.60	14,924	**24,786**

Precision, Recall, F-measure and Feedback are denoted respectively by P, R, F and Fb. The numbers of recommendations and queries are denoted respectively by #Rs and #Qs.

TABLE 5
ROSE RESULTS OF MINING REVISION HISTORIES IN [38]

	ROSE ($e = 1$)			
Project	P	R	F	Fb
Eclipse	0.29	0.36	**0.32**	0.80
GCC	0.35	0.59	**0.44**	0.76
GIMP	0.28	0.48	**0.35**	0.77
JBOSS	0.19	0.36	**0.25**	0.74
JEDIT	0.31	0.41	**0.35**	0.95
KOFFICE	0.30	0.45	**0.36**	0.87
POSTGRES	0.29	0.37	**0.33**	0.95
PYTHON	0.34	0.46	**0.39**	0.73
Average	0.29	0.44	**0.35**	0.82

[4] In a classification model, if the AUC value is less than 0.7, the model does not provide adequate discrimination. However, as our AUC metric was adjusted to calculate recommendation accuracy, it is not clear that the AUC value here has the same interpretation.

TABLE 6
MI-EA RESULTS FOR ACCURATE RECOMMENDATIONS

	MI-EA ($v = 3$, $e = 1$)					
Project	P	R	F	Fb	#Rs	#Qs
Mylyn	0.67	0.54	**0.59**	0.25	3,871	**15,248**
Platform	0.87	0.40	**0.55**	0.25	890	**3,543**
PDE	0.40	0.66	**0.50**	0.08	83	**988**
ECF	0.27	0.67	**0.38**	0.05	18	**352**
MDT	1.00	0.30	**0.46**	0.22	75	**336**
Others	0.78	0.37	**0.50**	0.16	696	**4,319**
Average	0.71	0.49	**0.58**	0.23	5,633	**24,786**

Precision, Recall, F-measure and Feedback are denoted respectively by P, R, F and Fb. The numbers of recommendations and queries are denoted respectively by #Rs and #Qs.

TABLE 7
MI-VO AND ROSE RESULTS FOR EARLY RECOMMENDATIONS

	MI-VO ($v = 3$, $e = 1$)			ROSE ($e = 1$)		
Project	F	N-th	#FRs	F	N-th	#FRs
Mylyn	0.18	**13th**	**1,300**	0.17	**24th**	**1,300**
Platform	0.23	**37th**	**158**	0.19	**56th**	**158**
PDE	0.18	**12th**	**94**	0.15	**20th**	**94**
ECF	0.09	**52nd**	**43**	0.10	**58th**	**43**
MDT	0.19	**25th**	**6**	0.23	**60th**	**6**
Others	0.24	**25th**	**143**	0.22	**57th**	**143**
Average	0.19	**17th**	**1,744**	0.18	**30th**	**1,744**

F-measure, the average of the first recommendation points, and the number of first recommendations are denoted by F, N-th, #FRs, respectively.

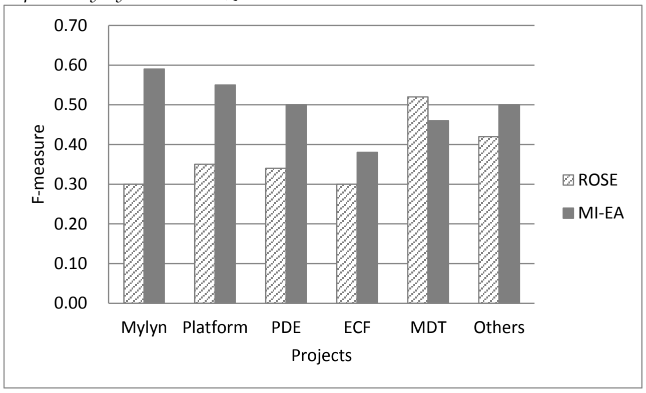


Fig. 6. Comparison of Recommendation Accuracy of MI and ROSE

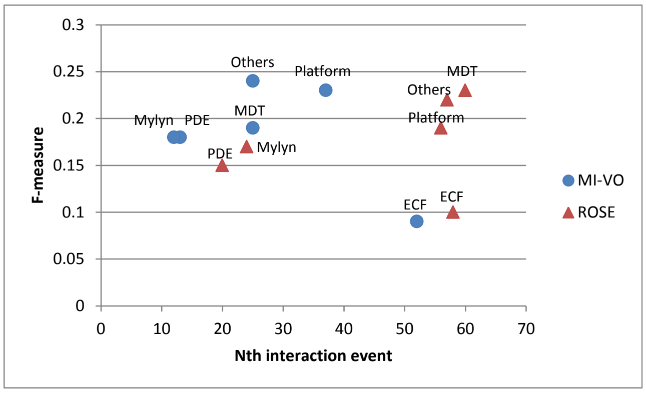


Fig. 7. Comparison of the First Recommendations of MI and ROSE

and 0.33 F-measure (F).

Our version of ROSE uses edited files in programmer interaction histories, while the original ROSE uses edited files in software revision histories. Our version of ROSE is used as a baseline for a fair comparison of techniques in our research questions. We then should know how these two approaches to implementing ROSE differ. Towards this, we extracted the original results of ROSE for coarse granularity from the paper [38] and calculated the F-measure from the original results.

Table 5 shows the original results. The average F-measure of ROSE in our experiment (Table 4) is 0.33, whereas the average F-measure of ROSE in the original experiment (Table 5) is also 0.35. As shown, the results for both versions of ROSE yield comparable results. The key differences are precision and recall. The precision average in Table 4 (0.43) is higher than that in Table 5 (0.29). The recall average in Table 4 (0.29) is lower than that in Table 5 (0.44). As noted in Section 4.3, there is a trade-off between precision and recall because the average size of the interaction traces used in our experiment is much larger than that of the change sets in the original experiment [38].

We use the ROSE recommendation results in Table 4 as baseline to evaluate the MI recommendation results (RQ2-4) in the following sections. Recall from Section 4.1 that for RQ2-4, we ran simulations by using three viewed files and one edited files (v = 3 and e = 1) as the context for MI.

4.2.3 RQ2: Can using viewed files create more accurate recommendations than using only edited files?

In Table 6 we show the results for MI-EA for each set of traces, as well as the average across each set. For example, in the Platform project, MI-EA recommended files to edit with 0.87 precision (P), 0.40 recall (R), and 0.55 F-measure (F). Consulting Table 4, we see the precision average value for MI-EA and ROSE are 0.71 and 0.41, respectively. The recall averages for MI-EA and ROSE are 0.49 and 0.28, respectively. The F-measure averages for MI-EA and ROSE are 0.58 and 0.33, respectively. We thus see that on average---both per set of traces and across all traces---MI-EA nearly always yields higher recommendation accuracy than ROSE.

We illustrate this in Fig. 6. The recommendation accuracy of MI-EA is consistently higher than that of ROSE across all of the projects except MDT. While the precision of MI-EA for MDT is still higher than that of ROSE (1.0 vs. 0.9), the recall of MI-EA for MDT is lower than that of ROSE (0.30 vs. 0.52). This may be related to the view-edit-ratio of a project. While other projects have view-edit-ratios ranging from 5.1 to 16.3 (Table 2), MDT exhibits a 77.9 view-edit-ratio, indicating that programmers viewed seventy eight files for each file edited. To better understand why MDT has such a high view-edit-ratio, we investigated the MDT traces and found that edit events which have different starting and ending times occurred in only 6 out of the 245 traces of the MDT project. In this case, three viewed files, as context information to recommend files to edit, are not very informative, thus reducing the likelihood of an effective recommendation.

To check the statistical significance of the results, we ran the Mann-Whitney U-test [2] with the 5,633 recommendation results of MI-EA and the 14,924 results of ROSE and compared the recommendation accuracy (F-measure) levels. Mann-Whitney U-test was selected as the data may not follow a normal distribution and thus a non-parametric statistical method is preferable. The resulting p-value was very close to 0.00, well below the traditional 0.05 level significance check. Considering the assumption of independence of the test, we reran the test with 6 F-measure values of MI in Table 6 and ROSE in Table 4, and found that the resulting p-value was again < 0.05. We also checked the practical significance by calculating the effect size [2] from the test. The effect size is 0.39, which is generally considered moderately strong. This indicates that the difference is not only statistically significant, but also practically significant.

In addition, Table 4 and 6 also present the feedback. The feedback average of MI-EA is 0.23, while that of ROSE is 0.60. The 0.23 feedback of MI-EA implies that about 2 out of 10 queries will create recommendations. The feedback of MI-EA is lower than that of ROSE. However, it was noted that MI can produce more recommendations than ROSE by creating recommendations at viewpoints. The low feedback is thus overcome by the more frequent queries of MI.

Given these results, we conclude:

Using viewed files (MI-EA) along with edited files yields significantly higher file-level recommendation accuracy than using only edited files (ROSE) for files to edit.

4.2.4 RQ3: Can using viewed files create earlier recommendations than using only edited files?

In Table 7, we show the time at which the first recommendations for MI-VO and ROSE are made, and the recommendation accuracies using F-measure. Table 7 presents the first recommendation point for both tools. For example, in the Platform project, MP-VO makes the first recommendation at the 37th timepoint on average, whereas ROSE makes the same at the 56th timepoint on average. Table 7 also presents the averages of all of the recommendation points. The average value for MI-VO is 17th and ROSE is 30th.

Fig. 7 compares the first recommendation points of MI and ROSE as well as their F-measure values shown in Table 7. Fig. 7 shows that MI-VO creates the first recommendation earlier than ROSE in all cases. With respect to recommendation accuracy, MI-VO shows higher F-measure values than ROSE for Mylyn, Platform, PDE and others, while showing lower F-measure values than ROSE for ECF and MDT. Finally, Table 7 presents the averaged recommendation performance. The F-measure value of MI-VO is slightly higher than that of ROSE (0.19 vs. 0.18).

MI-VO allows us to provide faster feedback to the user while still making recommendations that are as accurate as the first recommendations provided by ROSE.

Given these results, we conclude as follows:

Using viewed files (MI-VO) along with edited files produces earlier recommendations than using only edited files (ROSE) for files to edit while still maintaining recommendation accuracy.

4.2.5 RQ4: Can using viewed files create more flexible than using only edited files?

We counted the numbers of recommendations of MI-VOA, and measured its recommendation accuracy, shown as Table 8. We compare the recommendation results of MI-VOA in Table 8 with those of ROSE in Table 4.

For example, in the Platform project, MI-VOA creates 7,853 recommendations, while ROSE creates 1,974 recommendations. Table 8 also presents the total number of recommendations. MI-VOA makes 46,380 recommendations, compared to ROSE's 14,924 recommendations. MI-VOA produces more recommendations than ROSE.

In addition, Table 8 presents the recommendation performance. MI-VOA shows the highest F-measure (0.63), precision (0.77) and recall (0.53), significantly outperforming ROSE (0.33, 0.41, and 0.28).

Fig. 8 compares the averaged *m*-th recommendations of MI and ROSE. In Fig. 8, each point represents the average value of F-measure and the average number of the interaction events of all of the m-th recommendations.

Figs. 8 (a) to (d) show the recommendation results of project Mylyn, Platform, PDE and ECF, respectively. For example, in the Platform project, MI-VOA makes the first recommendation at the 24th timepoint with 0.35 F-measure, and the second recommendation at the 36th timepoint with 0.37 F-measure on average. On the contrary, ROSE makes the first recommendation at the 56th timepoint with 0.20 F-measure on average. The F-measure average of MI-VOA starts from 0.35, whereas that of ROSE starts from 0.2.

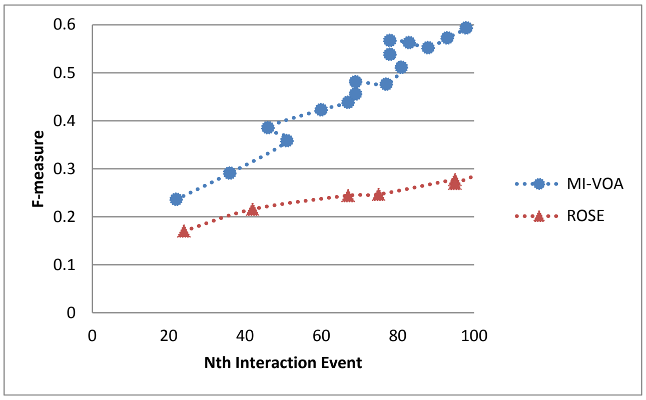
In overall, Fig. 8 shows that MI-VOA creates the *m*-th recommendation earlier than ROSE, with a higher overall recommendation accuracy. Furthermore, the gap between the *m*-th recommendation accuracies becomes larger as *n* increases across projects.

Fig. 8 also shows that MI-VOA creates more *m*-th recommendations than ROSE within the first 100 interaction events. MI-VOA produced 15 recommendations for the Mylyn projects, 10 for Platform, 8 for PDE, and 3 for ECF. ROSE produced 5 for Mylyn, 1 for Platform, 3 for PDE,

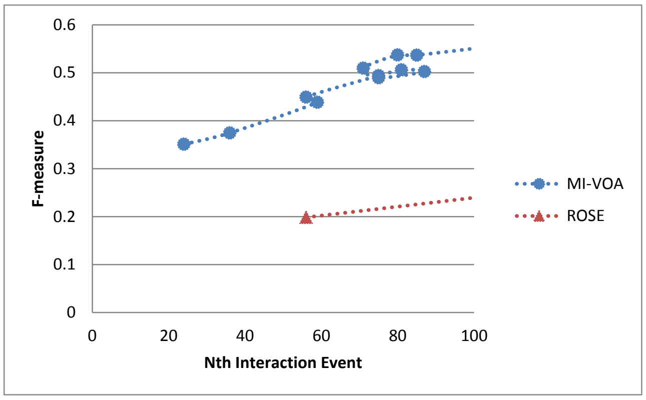
TABLE 8
THE RESULTS FOR FLEXIBLE RECOMMENDATIONS

	MI-VOA ($v = 3$, $e = 1$)					
Project	P	R	**F**	Fb	#**Rs**	#Qs
Mylyn	0.70	0.58	**0.63**	0.21	**24630**	117212
Platform	0.85	0.48	**0.61**	0.24	**7853**	32225
PDE	0.88	0.94	**0.91**	0.21	**1489**	6930
ECF	0.56	0.79	**0.66**	0.06	**199**	3305
MDT	1.00	0.37	**0.54**	0.29	**651**	2218
Others	0.85	0.43	**0.57**	0.36	**11558**	32211
Average	0.77	0.53	**0.63**	0.24	**46380**	280975

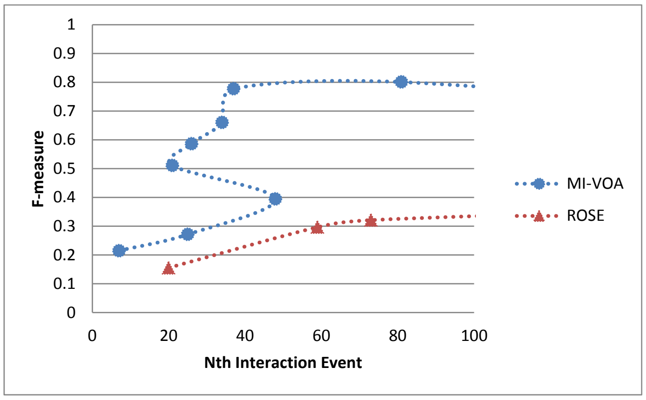
Precision, Recall, F-measure and Feedback are denoted respectively by P, R, F and Fb; The number of recommendations and the number of queries are denoted respectively by #Rs and #Qs.



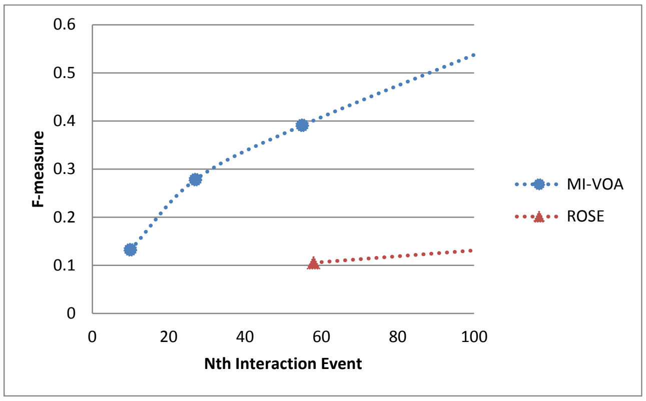
(a) Recommendations of the Mylyn Project



(b) Recommendations of the Platform Project



(c) Recommendations of the PDE Project



(d) Recommendations of the ECF Project

Fig. 8. Comparison of the Nth Recommendations of MI and ROSE

and 1 for ECF (3~15 vs. 1~3 recommendations).

Given these results, we conclude:

Using viewed files (MI-VOA) along with edited files produces more recommendations with higher accuracy than using only edited files (ROSE) for files to edit.

5 Discussion

Our experimental results demonstrated that MI yields higher recommendation accuracy than ROSE (0.63 vs. 0.35 F-measure). Our results also demonstrate that MI creates recommendations much earlier than ROSE, and also more frequently than ROSE. In this section, we explore issues raised by our study and issues for consideration in later work.

5.1 Reducing Recommendation Noise

To better understand how MI (v = 3, e = 1) yields significantly higher recommendation accuracy than ROSE (e = 1), we analyzed the recommendation results from the early recommendations produced in Section 4.2.4, as these are important in guiding programmer navigation. Furthermore, these early recommendations are one of the chief distinctions between MI and ROSE. We next selected examples that present a good contrast of the types of recommendations produced by MI and ROSE.

In the end, we selected seven recommendation results of MI and ROSE. Table 9 shows the selected recommendations. For example, for the bug report #256543 in the first row, MI created recommendations at the 3rd interaction with 1.0 precision and 1.0 recall, while ROSE at the 117th interaction with 0.4 precision and 1.0 recall. We examined the first recommendation result in interaction trace #123638 of bug report #256543 in Table 9.

For our selection, MI does not recommend unnecessary files to edit, while ROSE does. In interaction trace #123638, MI recommended the four files that are actually edited in the interaction trace {ISimplePropertyListener.java, StyledTextObservableValueDefaultSelectionTest.java, BindingTestSuite.java, TextObservableValueDefaultSelec-tionTest.java}. ROSE recommended these four files, as well as six other files that were unrelated to the interaction trace, yielding ten files in total and 0.4 precision.

To understand why, we then examined the contexts that MI and ROSE used. To generate the recommendation above, MI used a context that consists of three viewed files {Snipet008ComputedValue.java, WidgetValueProperty.java, SWTVetoableValueDecorator.java} and one edited file {SWTObservables.java}, while ROSE used a context that included only one edited file {SWTObservables.java}. This is considerable increase in the information available, resulting in more precise association rules.

This pattern---where MI has access to additional information over ROSE---is consistent throughout, and is what results in the consistent improvements in accuracy. This matches the initial intuition underlying this work, and demonstrates that the observed improvements are not simply due to another overlooked factor.

TABLE 9
THE RECOMMENDATION RESULTS OF MPI AND ROSE TO BE USED FOR THE QUESTIONNAIRE SURVEY

Project	#Bug Report	#Interaction Trace	MI			ROSE		
			P	R	N-th	P	R	N-th
Platform	256543	123638	1.0	1.0	3	0.4	1.0	117
	259411	132096	1.0	1.0	4	0.5	0.71	156
	291215	149633	1.0	0.42	4	0.1	0.04	23
PDE	239494	121439	1.0	0.91	5	0.2	0.2	95
Mylyn	280973	139683	1.0	1.0	3	0	0	59
	283093	148595	1.0	1.0	9	0.1	0.17	49
	258717	122750	0.4	0.8	4	0.2	0.29	228

The bug report number and the interaction trace number are identified by the Eclipse Bugzilla system [9]. Precision, Recall and the average of the first recommendation points are denoted by P, R, and N-th, respectively.

5.2 Users' Qualitative Evaluation

In Section 4.2, we demonstrated that MI produces recommendations more quickly and with higher recommendation accuracy than ROSE in most cases, and in Section 5.1, we found that MI produces less noisy recommendations than ROSE. While these results are encouraging and our evaluation has been conducted as typical for recommendation systems, the best approach for recommending files to edit is that best appeals to actual people in the end. To understand people's choices, we constructed a questionnaire survey comparing recommendations produced by MI and ROSE.

Using the seven recommendation results in Table 9, we created seven comparison questions. The questions ask the subject to choose between two recommendation results of ROSE and MI. To understand why the participants made their choices, our survey investigated their rationales with an essay question, "Would you give your reasons why you selected recommendation A or B?"

For this evaluation, we recruited fifteen participants, seven graduate students and eight industry developers. The graduate students are identified with the initial 'P0' followed by a number (i.e. P01~P07), and the industry developers are identified with the initial 'P1' followed by a number (i.e. P11~P18).

As we anticipated, the participants chose MI 86 times and ROSE 19 times. To understand why they made their choices, we examined the participants' rationales. Four participants indicated they preferred recommendations for files actually edited in practice (P1, P02, P06, and P14), while two stated recommending edited files is crucial to recommendation effectiveness (P03 and P12). Participant P12 stated, "This recommendation system should give programmers a hint. More important than giving accurate hints is to limit the points to focus on. Thus, fewer suggestions with a high hit rate are important." Similarly, five participants stated they favored recommendations including the files edited in practice, assuming that when working on an unsolved bug report, that they must visit files recently edited (P04, P05, P07, P15 and P16). For example, participant P16 stated, "I just want to start at some point which seems directly related to the issue." From these participants' answers, we infer that recommending small sets of files which include files that should actually be edited is strongly preferred by users.

In contrast, some participants wanted to see more recommendation items (P13 and P17). Participant P13 stated, "When the software system is tested well, it seems good that the tool makes a decision what to show. However, regarding unknown bugs, it is good to show as much information as possible and let programmers to make a decision." P17 stated, "It is better to consider all source codes that can have an impact." It is noted that P17 chose the ROSE recommendations three times, more than all but one other participant (P11). This may be because ROSE typically recommended more files than MI; the fact that the recommendations do not include the files edited in practice was viewed as a non-issue by P17. The participant, who chose the ROSE recommendations four times (P11) stated (somewhat enigmatically), "A software developer's decision differs from a tool's decision due to personal preference. My decision depended on the situation."

In general, however, we conclude that the aspects most valued by users are recommendations which include directly the files that need to be edited, and which limit the number of files recommended. We noted that several participants stated that reducing the number of irrelevant files recommended is a key consideration for them in using a recommendation system.

5.3 File-level Recommendation versus Method-level Recommendation

In this paper, we presented the recommendation results at the file level, not the method level. However, we have conducted the simulation in Section 4 at the method level, and found that recommendations at both levels produce nearly the same results.

Fig. 9 shows the recommendation results at the method level. Curves "*n*-views," "*n*-edits," and "*n*-views and *n*-edits" represent the recommendation results of using viewed methods ($v = n$), edited methods ($e = n$) and both methods ($v = n$ and $e = n$) as a context. When n is 1, the "*1*-view" yields 0.2 precision and 0.2 recall, the "*1*-edit" yields 0.6 precision and 0.2 recall, and "*1*-view and *1*-edit" yields 0.8 precision and 0.3 recall. The points of each curve go up as n increases. Fig. 9 shows that "*n*-views and *n*-edits" achieves consistently higher precision and recall than "*n*-edits."

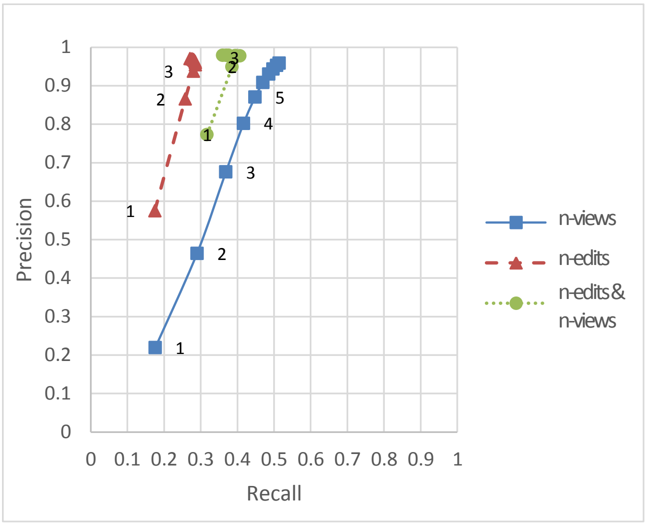


Fig. 9. Precision and recall graph at the method level

The method-level recommendation results were consistent with the file-level results: using additional viewed methods (MI) yields consistently higher recommendation accuracy compared to the use of only edited methods (ROSE). However, there was a slight difference between method-level recommendations and file-level recommendations. The precision at the method level is higher than the precision at the file level, while the recall at the method level is lower than the recall at the file level. This is because of the trade-off mentioned in Section 4.1.5 that arises when the number of viewed and edited methods is larger than the number of viewed and edited files.

For the purpose of the presentation, we felt file-level recommendations were considerably more simple and easier. Given the similarity of the simulation results, we therefore chose to show only the recommendation results at the file level.

5.4 View Contexts versus Edit Contexts

When we investigated the file-level recommendations for the Platform project, we found that edit contexts occasionally produce earlier recommendations than view contexts. For example, in the interaction trace #94360 of bug report #224588, the edit context (e = 1) made the first recommendation at the first edit event after one view event. The edit context (e = 1) yielded 0.6 precision by recommending two correct files {PatchMessages.java, PreviewPatchPage2.java} with three incorrect files. The view context (v = 3) made this recommendation at the 24th view event, after 8 edit events and yielded only 0.33 precision (one correct file {PatchMessages.java} and two incorrect files).

We found this result surprising—the intent of MI is to produce, at a minimum, earlier recommendations (along with more accurate recommendations). Upon examination, we found that the view context works better when programmers view first, while an edit context works better when programmers edit first. Typically speaking, programmers look, then edit. However for some tasks—simple bug fixing tasks, for example—programmers can often begin editing immediately, while for enhancements programmers must first understand code by viewing it [36].

Thus we see view context and edit context are complementary. Our proposed method leverages this by flexibly forming a context that reflects the complementary nature of view context and edit context, as with MI-VOA.

TABLE 10
THE RECOMMENDATION RESULTS OF THE PLATFORM PROJECT OBTAINED BY CHANGING THE MINIMUM SUPPORT

	ROSE (e = 1)		MI-VOA
	mSup = 1	mSup = 3	(v=3, e=1)
P	0.60	0.64	0.85
R	0.26	0.22	0.48
F	0.36	0.33	0.61
Fb	0.56	0.24	0.24
#Recommendations	1,974	843	7853
#Queries	3,543	3,543	32,225

5.5 Feedback and Recommendation Accuracy

In Section 4.2.2, we noted the feedback of MI is lower than that of ROSE (0.23 vs. 0.60). As a lower feedback can in some cases result in a higher accuracy, we wished to determine if this was the case here. Towards this, we reran the simulation using a higher minimum support (the minimum number of co-occurrences of the antecedent and the consequent of an association rule) for ROSE (moving from one to three).

In doing so, we lowered the feedback of ROSE, allowing for a comparison against MI in which the feedback was comparable. This is shown in Table 10: by moving to a minimum support of 3, rather than 1, the feedback for ROSE is lowered from 0.56 to 0.24, mirroring the 0.24 achieved by MI-VOA. As expected, this positively impacts the precision for ROSE, moving from 0.60 to 0.64, but the precision is still considerably less than the 0.85 precision achieved by MI-VOA. (Recall is also reduced for ROSE, dropping from 0.26 to 0.22. This is also substantially less than the 0.48 achieved by MI-VOA.)

We interpret this as showing that MI yields higher recommendation accuracy than ROSE even when feedback values are similar, and that the positive results relative to ROSE seen in Section 4.2 are not due to lower feedback.

5.6 Acceptable level of Recommendation Accuracy

By referring to Fig. 4, we can suggest how many viewed and edited files are necessary to achieve an acceptable (or even higher) level of recommendation accuracy. For this discussion, we assume that a level of recommendation accuracy is acceptable when more than half of recommendations are correct. In other words, the precision is higher than 0.5. Fig. 4 shows that when n is smaller than 5, a view and edit combined context (v = n and e = n) produce higher precision and recall than a view only context (v = n) or an edit only context (e = n), with the peak recall occurring at view and edit contexts of (v = 3 and e = 3). When n is larger than 5, the view context (v = 5) produces higher precision and recall than the edit context (e = 1) and the view and edit context (v = 1 and e = 1). Therefore we suggest using a context that includes less than or equal to 3 edited files with more than 5 viewed files as acceptable levels of file-level recommendation accuracy. However, this is a rough conclusion based on the results of the cases where the numbers of viewed files and edited files are equal ($v = e = n$). To give conclusive numbers, more thorough investigations are needed.

Throughout this paper, we showed that our approach MI better predicts where a developer will edit than ROSE, yet did not evaluate if it predicts where the developer will need to edit. We leave this as future work, expecting that new approaches and evaluations will focus on the locations that ought to be edited.

6 Threats to Validity

Internal validity. We conducted a simulated comparative controlled experiment, where the only change was the inclusion of viewed files, with other conditions identical. As MI was an extension of ROSE, the difference in techniques was managed by using a software flag, with most of the implementation between the techniques identical. We thus believe the results of Section 4.2 are indeed due to the inclusion of viewed files. For the human study, each subject was given the same choices, and the order of recommendation (ROSE vs. MI) was random and anonymized to prevent this from biasing the results.

Construct validity. We used precision, recall, F-measure, ROC and AUC to measure the recommendation accuracy, and used a questionnaire survey to evaluate by humans. The measurements used are typical measurements used in recommendation system research, and the use of several measurements and evaluations—all indicating the same conclusion—mitigate construct validity threats.

Conclusion validity. We used the Mann-Whitney U-test to determine the statistical significance of the difference between the recommendation results of MI and those of ROSE. When the number of observations is very high (as was the case here), the Mann-Whitney U-test may conclude that the two results are significantly different, when in fact the practical difference is very small. To compensate for this, we also calculated the effect size to measure the practical significance. The effect size indicates that the recommendation accuracy, as improved by MI, has practical significance.

External validity. We used the interaction traces of 72 Eclipse sub-projects, which may not be representative of all software projects. Also, our experimental results are consistent with the projects which have the view-edit ratio from 5.1 to 16.3, while inconsistent with the MDT project which has a 77.9 view-edit ratio. Therefore, our results may be generalized to the class of open projects that use the Eclipse Bugzilla system and have a view-edit ratio in the range of 5.1~16.3.

7 Conclusion

In this work, we have examined how the use of view information, gathered from programmer interaction histories, can help provide a more detailed context of programmer activity leading to more accurate, earlier and more flexible edit recommendations. To evaluate this, we replicated the previous approach ROSE and proposed a new approach MI, which extends ROSE by additionally considering the records of viewed files. We then conducted a simulated comparative controlled experiment by mining the records of files that programmers had both viewed and edited (MI), and mining the records of files that programmers had only edited (ROSE). In this experiment, we found that MI recommends files to edit with 58~63% accuracy while ROSE does 33~35% accuracy. MI also creates recommendations earlier than ROSE (17th vs. 30th interaction event) and more flexibly (3~15 vs. 1~3 recommendations within the first 100 interaction events).

Overall, our work makes the following contributions:

- We developed a new powerful context formation approach and demonstrated its efficacy in mining programmer interaction histories (MI).
- We developed a comparative framework that helps understand which factors have much influence on the recommendation performance.
- We demonstrated that the rules mined by our approach (MI) outperform the rules mined from edit histories (ROSE) [38] in recommending files to edit.
- We identified that the significant improvement of the recommendation performance is enabled by the context further elaborated by the records of files viewed.

Our research can be utilized in several ways. First, our results provide basic evidence that leveraging the detailed records of what programmers viewed will improve mining approaches for recommending various other artifacts (e.g., documents [6], test sets [4], and e-mails [3]). Second, our proposed context formation can be a basis for developers to flexibly adapt the context information in a recommendation system to yield better recommendation results whenever the user situation changes. In short, we have demonstrated that as the context information considered increases to better reflects user behavior, the system's behavior will better reflect the user's needs.

Acknowledgment

This work was supported under the framework of international cooperation program managed by National Research Foundation of Korea (NRF-2013K2A1A2055116), and was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2013R1A6A3A 01060158). This research was also supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1020) supervised by the NIPA (National IT Industry Promotion Agency).

References

[1] R. Agrawal, T. Imielinski and A. N. Swami, "Mining association rules between sets of items in large databases," *Proc. ACM*

SIGMOD International Conf. on Management of Data, Washington, D.C., May 26-28, 1993, ACM Press, pp. 207–216.

[2] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," *Proc. 33rd International Conf. on Software Engineering (ICSE '11)*, ACM, New York, NY, USA, pp. 1-10.

[3] A. Bacchelli, M. Lanza and B. Humpa, "RTFM (Read the Factual Mails) -augmenting program comprehension with remail," *Proc. 15th IEEE European Conf. on Software Maintenance and Reengineering (CSMR '11)*, IEEE CS Press, pp. 15–24.

[4] A. Begel, Y. P. Khoo and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," *Proc. 32nd ACM/IEEE International Conf. on Software Engineering*, Vol. 1, 2010, pp. 125–134.

[5] G. Canfora, M. Ceccarelli, L. Cerulo and M. Di Penta, "Using multivariate time series and association rules to detect logical change coupling: An empirical study," *Proc. IEEE International Conf. on Software Maintenance (ICSM)*, 2010, no. i, pp. 1–10.

[6] D. Cubranic and G. C. Murphy, "Hipikat: recommending pertinent software development artifacts," *Proc. 25th International Conf. on Software Engineering (ICSE '03)*, IEEE Computer Society, Washington, DC, USA, 2003, pp. 408-418.

[7] R. DeLine, M. Czerwinski and G. Robertson, "Easing program comprehension by sharing navigation data," *Proc. IEEE Symposium on Visual Languages and Human Centric Computing*, 2005, pp. 241-248.

[8] A. K. Dey, "Understanding and using context," *Personal Ubiquitous Computer,* 5(1), Jan. 2001, pp. 4-7.

[9] Eclipse Bugzilla, https://bugs.eclipse.org/bugs/.

[10] T. Fawcett. 2006. An introduction to ROC analysis. *Pattern Recogn. Lett.* 27, 8 (June 2006), pp. 861-874.

[11] B. Fluri, M. Wuersch, M. Pinzger and H. Gall, "Change distilling: tree differencing for fine-grained source code change extraction," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, 2007, pp. 725-743.

[12] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.

[13] L. Hattori, M. Lungu and M. Lanza, "Replaying past changes in multi-developer projects," *Proc. Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (IWPSE-EVOL '10)*, ACM, NY, USA, 2010, pp. 13-22.

[14] L. Hattori, M. D. Ambros, M. Lanza and M. Lungu, "Software evolution comprehension: replay to the rescue," *Proc. 19th International Conf. on Program Comprehension (ICPC '11),* IEEE Computer Society, Washington, DC, USA, 2011, pp. 161-170.

[15] F. Jaafar, Y.-C. Gueheneuc, S. Hamel and G. Antoniol, "An exploratory study of macro co-changes," *Proc. 18th Working Conf. on Reverse Engineering (WCRE '11),* IEEE Computer Society, Washington, DC, USA, 2011, pp.325-334.

[16] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," *Proc. 33rd International Conf. on Software Engineering (ICSE '11),* ACM, New York, NY, USA, 2011, pp. 351-360.

[17] M. Kersten, and G. C. Murphy, "Using task context to improve programmer productivity," *Proc. 14th ACM SIGSOFT international symposium on Foundations of software engineering (SIGSOFT '06/FSE-14),* ACM, New York, NY, USA, pp. 1-11.

[18] D. Kim, Y. Tao, S. Kim and A. Zeller, "Where Should We Fix This Bug? A Two-Phase Recommendation Model," *IEEE Transactions on Software Engineering*, vol.39, no.11, Nov. 2013, pp. 1597-1610.

[19] M. Kim and D. Notkin, "Discovering and representing systematic code changes," *Proc. 31st International Conf. on Software Engineering (ICSE '09),* IEEE Computer Society, Washington, DC, USA, 2009, pp. 309-319.

[20] N. Littlestone, "Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm," *Machine Learning*, Kluwer Academic Pub. 1988, pp. 285-318.

[21] S. Lee, S. Kang and M. Staats, "NavClus: A graphical recommender for assisting code exploration," *Proc. 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 1315-1318.

[22] S. Lee and S. Kang, "Clustering navigation sequences to create contexts for guiding code navigation," J. Syst. Softw. 86, 8, August 2013, pp. 2154-2165.

[23] T. Lee, J. Nam, D. Han, S. Kim and H. P. In, "Micro interaction metrics for defect prediction," *Proc. European Software Engineering Conf. and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011)*. Szeged, Hungary, September 5-9, 2011.

[24] J. McGrath, Methodology matters: doing research in the behavioral and social sciences, Oct. 1994, pp. 162-189.

[25] C. Parnin, C. Görg and S. Rugaber, "Enriching revision history with interactions," *Proc. international workshop on Mining Software Repositories (MSR '06),* ACM, NY, USA, 2006, pp. 155-158.

[26] C. Parnin and C. Görg, "Building usage contexts during program comprehension," *Proc. 14th IEEE International Conf. on Program Comprehension (ICPC '06),* IEEE Computer Society, Washington, DC, USA, 2006, pp. 13-22.

[27] D. Piorkowski, S. D. Fleming, C. Scaffidi, L. John, C. Bogart, B. E. John, M. Burnett and R. Bellamy, "Modeling programmer navigation: A head-to-head empirical evaluation of predictive models," Proc. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC),* IEEE, 2011, pp. 109–116.

[28] D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy and C. Swart, "Reactive information foraging: an empirical investigation of theory-based recommender systems for programmers," Proc. *ACM annual conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1471–1480.

[29] R. Robbes and M. Lanza, "Characterizing and understanding development Sessions," *Proc. 15th IEEE International Conf. on Program Comprehension (ICPC '07),* IEEE Computer Society, Washington, DC, USA, 2007, pp. 155-166.

[30] R. Robbes, D. Pollet and M. Lanza, "Logical coupling based on fine-grained change information," *Proc. of 15th IEEE Working Conf. on Reverse Engineering (WCRE '08),* IEEE CS Press, 2008, pp. 42-46.

[31] R. Robbes, D. Pollet and M. Lanza, "Replaying IDE interactions to evaluate and improve change prediction approaches," *Proc. Working Conf. on Mining Software Repositories,* 2010, pp. 161-170.

[32] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, 2010, pp. 80-86.

[33] M. P. Robillard and B. Dagenais, "Recommending change clusters to support software investigation: an empirical study," *Journal of Software Maintenance and Evolution: Research and Practice*, 2010, pp. 143-164.

[34] N. Sawadsky, G. C. Murphy and R. Jiresal, "Reverb: Recommending code-related web pages," Proc. of the 35th ACM/IEEE Int'l Conf. on Software Engineering, 2013.

[35] J. Singer, R. Elves and M. A. Storey, "NavTracks: supporting navigation in software maintenance," *Proc. 21st IEEE International Conf. on Software Maintenance (ICSM '05),* IEEE Computer Society, Washington, DC, USA, pp. 325-334.

[36] A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Transactions on Software Engineering*, 30, 2004, pp. 574–586.

[37] A. T. T. Ying and M. P. Robillard, The influence of the task on programmer behavior. *Proc. 19th IEEE International Conf. on Program Comprehension (ICPC '11),* June 2011.

[38] T. Zimmermann, P. Weissgerber, S. Diehl and A. Zeller, "Mining version histories to guide code changes," *IEEE Transactions on Software Engineering*, 31(6), 2005, pp. 429–445.

Seonah Lee received her BS and MS in Computer Science and Engineering from Ewha Womans University (1993 to 1999). She worked as a software engineer in Samsung Electronics (1999 to 2006). She also received MSE in School of Computer Science from Carnegie Mellon University (2004 to 2005). She was a PhD student at School of Computer Science, University of British Columbia (2006 to 2009) but left. She received her PhD in School of Computer Science, KAIST (2010 to 2013). She is currently a postdoctoral researcher at KAIST. Her research interest includes software evolution, program comprehension, code recommendation, data mining, software visualization, and software architecture.

Sungwon Kang received his BA from Seoul National University, Korea, in 1982 and received his MS and PhD in computer science from the University of Iowa, USA, in 1989 and 1992, respectively. From 1993, he was a principal researcher of Korea Telecom R & D Group until October 2001 when he joined KAIST and is currently an associate professor of the university. Since 2003, he has been an adjunct faculty member of Carnegie-Mellon University, USA, for the Master of Software Engineering Program. He is the editor of Korean Journal of Software Engineering Society. His current research areas include software architecture, software product line, software modeling and analysis, and software testing.

Sunghun Kim received the PhD degree from the Department of Computer Science, University of California, Santa Cruz, in 2006. He is an assistant professor of computer science at the Hong Kong University of Science and Technology. He was a postdoctoral associate at the Massachusetts Institute of Technology and a member of the Program Analysis Group. He was the chief technical officer (CTO) and led a 25-person team for six years at the Nara Vision Co. Ltd., a leading Internet software company in Korea. His core research area is software engineering, focusing on software evolution, program analysis, and empirical studies. He is a member of the IEEE.

Matt Staats received his PhD from the Department of Computer Science and Engineering, University of Minnesota Twin-Cities, in 2011. He was a postdoctoral researcher at KAIST from 2011 to 2013, and a research fellow at the University of Luxembourg from 2013 to 2014. Matt Staats conducted this work at the Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg. He is currently employed by Google, Inc.