Improving Change Impact Prediction by Aggregating Source Code Changes

Anonymous authors
Paper under double-blind review
Anonymous University

***Abstract*—Changes on source code may propagate to distant code entities through various kinds of relationships, causing corresponding changes to ensure software system consistency. It is difficult for developers to find the potentially impacted code entities at a distance among the sophisticated interrelationships in source code. In this work, we propose a novel approach called IMPPRED that predicts the impact of code changes by combining evolutionary coupling and spatial-temporal change relationships (such as structural dependencies and code clone relationships) among code entities. Furthermore, we propose a change impact prediction ranking method with the combination of association rule mining and spatial-temporal change analysis. We evaluate IMPPRED with six high-quality open-source Java projects, involving evolution histories of totally 19,003 commits. Our evaluation shows that IMPPRED outperforms the state-of-the-art methods with improvements up to 22% in mean average precision (MAP) and 18% in recall. The Top-5 effectiveness is also improved, with increases by 24% in MAP, 11% in precision, 24% in recall, and 15% in F1-score. Moreover, the result of sensitivity analysis confirms the effectiveness of the configurable parameters. The results of applicability analysis and efficiency analysis further confirm the practicality of IMPPRED.**

***Index Terms*—Change Impact Prediction, Evolutionary Coupling, Change Impact Analysis, Spatial-Temporal Change Relationship**

I. INTRODUCTION

Software systems often require changes to their constituent code entities, such as classes, member variables, and member methods, for delivering new products and services. However, developers face an increasingly challenging task in locating the impact of these changes on the software system. The difficulties in this task is subject to the sophisticated direct and indirect relationships, e.g., evolutionary coupling [1], [2], structural dependencies [3] and code clones [4], [5]. Changes to one entity may result in changes to multiple related entities [6], which must be made to maintain the consistency of the software system [7], [8]. Failure to properly locate and modify related code entities can result in software errors or inconsistencies [9]. To address these challenges, developers require methods to predict potentially impacted code entities, which is of great importance for developers to improve their development quality and efficiency.

Various techniques have been proposed to predict the impact of code changes by association rule mining [10]–[13], code change pattern mining [7], [9], [14], and machine learning [15], [16]. Association rule mining [10], [17] is a most widely used method, which relies on the evolutionary coupling [1], [2] relationship among code entities. However, this method may result in low recall or even totally not applicable if the source code entities seldom co-change or never co-changed at all [11], [18]. Code change pattern mining or machine learning-based method are suitable for finding specific change patterns (e.g., co-change patterns [14], [15], [19], defect repair patterns [20]) or perform well on specific domains (e.g., consistency-maintenance for code clones [16], defect prediction [21], [22]). However, the number of patterns or scenarios they cover may be limited, which will limit the applicability [11] of these methods.

Some researchers also proposed techniques to enhance the applicability of the impact prediction, such as TAR [12], aggregated association rules [23], and adaptive recommendation [18]. There are also many attempts to improve the recall of potentially impacted code entities, including concept coupling [24], [25], applying structural dependencies [3], [26], combining code clones [4], [5], [27], and considering requirement relationships [28]. Although these methods exhibited an increase in recall, the lack of effective prioritization strategy restricted their ability to provide practical assistance to developers during the development process. Hence, further research is needed to improve the change impact prediction method, which requires both high recall (complete list of change impacts) and precise ranking information (prioritized list of change impacts).

In this work, we propose a novel code change impact prediction method based on change aggregation, called IMPPRED. It effectively captures the impacted code entities by combining the evolutionary coupling, i.e., association rules, and spatial-temporal change relationships (such as structural dependencies and code clone relationships) capturing changed code entities and their relationships within a limited time and space range, thus improves the completeness of the prediction results. Moreover, we propose a change impact prediction ranking method with the combination of association rule mining and spatial-temporal change analysis to improve the effectiveness of priority ranking for the prediction results.

The proposed method is evaluated through experiments conducted on six high-quality open-source Java projects, which involving evolution histories of totally 19,003 commits. Our evaluation shows that IMPPRED outperforms the state-of-the-art methods with improvements up to 22% in MAP and 18% in recall. The Top-5 effectiveness is also improved, with increases by 24% in MAP, 11% in precision, 24% in recall, and 15% in F1-score. Moreover, the result of sensitivity analysis confirms

the effectiveness of the configurable parameters. The results of applicability analysis and efficiency analysis further confirm the practicality of IMPPRED. In summary, this paper presents three main contributions:

- We propose a code change impact prediction method named IMPPRED based on change aggregation to improve the completeness of prediction results.
- We propose a candidate result ranking method to improve the effectiveness of priority ranking for prediction results;
- We construct a code change impact prediction dataset and validate the effectiveness and practicality of IMPPRED.

The rest of the paper is organized as follows. Section II introduces the basic concepts and definitions. Section III formulates the research problem. Section IV details our approach. We discuss the process and results of the evaluation in Section V and Section VI. Section VII discusses the threats to validity. Finally, we discuss related work in Section VIII before a conclusion in Section IX.

II. CONCEPTS AND DEFINITIONS

A. Evolutionary Coupling

Evolutionary coupling is a relationship among code entities that are frequently changed together [2], [29]. It is commonly expressed by **association rule**s [10], [17] in the form of $A \Rightarrow B$, where A and B are sets of code entities. The notion $A \Rightarrow B$ implies, if A is changed, then B is also likely to be changed [5], [10], [30]. The likeliness of this association rule is measured by the **support** and **confidence**. The support of A is the number of commits in which all code entities in A changed together, and the support of $A \Rightarrow B$ is the frequency of changes to both A and B. We define $support(A \cup B) = support(A \Rightarrow B) = support(B \Rightarrow A)$ as they indicate the frequency of co-changes between A and B [10]. The confidence of the rule, $confidence(A \Rightarrow B)$, is the ratio of co-changes of A and B to all changes of A. Suppose that A changed in the commits 2, 3, 5, 9, 11, and B changed in the commits 1, 3, 4, 9, 11, 12. Thus, $support(A) = 5$, $support(B) = 6$, and $support(A \cup B) = support(A \Rightarrow B) = support(B \Rightarrow A) = 3$, $confidence(A \Rightarrow B) = 3/5$ and $confidence(B \Rightarrow A) = 3/6$.

Targeted association rule mining is an approach for change impact analysis (CIA) that uses association rules mined from software change history [10], [11], [31], [32]. The change history $\mathcal{H}$ is considered a *collect of transactions*, with each transaction being a commit of changed code entities. Given a set of *initial changed code entities*, called a **query** Q and the change history $\mathcal{H}$, this method filters the relevant transactions and then uses association rule mining to obtain relevant rules, thereby predicting the impacted code entities [11], [18]. TARMAQ [11] employs targeted association rule mining to suggest code changes. Its effectiveness has been demonstrated in various studies [5], [18], [23], [33]. Another popular approach is ROSE [10], which also uses such technique, but differs in its selection of relevant transactions. TARMAQ considers a transaction T relevant only if it intersects with the query Q, while ROSE requires that the query Q is a subset of the transaction T.

B. Code Entity Genealogy

During the software evolution process, a specific code entity may be created in a particular commit and remains active in subsequent commits. For each code snapshot corresponding to these commits, there exists a version of the code entity in the snapshot. The code entity genealogy represents all snapshot versions of a code entity, which includes *class genealogy*, *variable genealogy*, and *method genealogy*.

C. Software Relationship Graph

In this work, we utilize twelve types of code entity relationships, including ten types of structural dependencies [22], [34], the structural containing relationship, and the code clone relationship. These relationships are chosen based on previous research showing their effectiveness in conveying change or bug propagation [3], [35], [36]. We define the Software Relationship Graph (SRG) to describe these software relationships at the code entity level and create a comprehensive dataset of SRGs by constructing one for each commit.

Definition 1: The SRG is a directed graph, $SRG = (V, E)$, where the vertices (V) are code entities and the edges (E) are the relationships between them. The vertices are labeled with their corresponding code entity types, such as Class, Method, and Variable, while the edges are labeled with the type of relationship, such as EXT, IMPL, CALL, etc. Multiple edges between two vertices are permitted if more than one relationship exists between them.

Table I shows the list of the relationship types. Specially, according to previous research by Mondal et al. [5], clone relationships comprising code fragments of three or more lines can aid in CIA. Therefore, we extend our analysis to include code clone at the fragment level between methods, with a minimum code size of three lines for fragment clones. The threshold for code clone similarity is set at the typical value of 0.7 based on previous studies [37], [38].

III. PROBLEM AND MOTIVATION

This section discusses the limitations of the targeted association rule mining technique for change impact prediction, and then motivates our approach with an example.

A. Problem Description

According to the definition of applicability [11], a targeted association rule mining technique $\mathcal{T}$ is only applicable to the query Q if there exists at least one transaction T that satisfies both of the following criteria: a) $A = T \cap Q \neq \emptyset$; b) $B = T - Q \neq \emptyset$. Targeted association rule mining is a widely used technique for change impact prediction [10]–[13], while there exists three *limitations*: 1) *Applicability problem*, which is mainly caused by the failure to meet the condition a) or b). In other words, if the potentially-impacted code entity had not been modified together with any code entities in Q in the history, the prediction will fail. This is referred to as

TABLE I
THE TWELVE TYPES OF CODE ENTITY RELATIONSHIPS

Type	Abbr.	Description
Extend	EXT	If class *A* inherits from class *B* via keyword "*Extends*", then there is a directed edge from class *A* to class *B* to denote their EXT relationship.
Implements	IMPL	If class *A* realizes interface *B* via keyword "*Implements*", then there is a directed edge from class *A* to class *B* to denote their IMPL relationship.
Member Variable	MVAR	If class *A* declares a member variable *b* with type of class *B*, then there is a directed edge from variable *b* to class *B* to denote their MVAR relationship.
Access	ACC	If one method *A.m* or member variable *A.a* accesses another member variable *A.b or B.b*, then there is a directed edge from method *A.m* or variable *A.a* to variable *A.b or B.b* to denote their ACC relationship.
Call	CALL	If one method *m* calls another method *n*, then there is a directed edge from method *m* to *n* to denote their CALL relationship.
Method Override	MOVR	If one method *m* overrides another method *n*, then there is a directed edge from method *m* to *n* to denote their MOVR relationship.
Parameter	PAR	If method *m* has at least one parameter with type of class *B*, then there is a directed edge from method *m* to class *B* to denote their PAR relationship.
Create	CRE	If class *B* is created in method *m* or member variable *b*, then there is a directed edge from method *m* or variable *b* to class *B* to denote their CRE relationship.
Local Variable	LVAR	If method *m* declares a local variable with type of class *B*, then there is a directed edge from method *m* to class *B* to denote their LVAR relationship.
Return Type	RET	If method *m* has a return type of class *B*, then there is a directed edge from method *m* to class *B* to denote their RET relationship.
Contain	CON	If method *m* or variable *a* is the member of class *A*, then there is a directed edge from class *A* to method *m* or variable *a* to denote their CON relationship.
Clone	CLO	If a clone relationship exists between the code fragments to which two methods belong, a bidirectional edge is established to denote their CLO relationship.

co-change history missing. 2) *Incomplete impact prediction*, which is also caused by co-change history missing. 3) *Lack of effective ranking strategy* [33], resulting that it is difficult to determine which ranking strategy to choose for the prediction.

B. Motivation Example

Fig. 1 illustrates our research motivation. Given the initial query $Q = \{m_1, m_2\}$ at commit c_{10}, the expected prediction of the change impact set is $\{m_3, m_4, m_6\}$. Each commit c_i has a corresponding transaction T_i, which represents the list of methods changed in the commit, i.e., $T_4 = \{m_1, m_2, m_5\}$, $T_5 = \{m_1, m_2, m_4, m_5\}$, $T_6 = \{m_3, m_6\}$. We assumed that the time span covering commits c_4, c_5, c_6 is about 12 hours.

Here, if we use TARMAQ to predict change impact, filtering relevant transactions to obtain $\{T_4, T_5\}$. For T_4, an association rule was created as $\{m_1, m_2\} \Rightarrow m_5$, and for T_5, two rules were created: $\{m_1, m_2\} \Rightarrow m_4$ and $\{m_1, m_2\} \Rightarrow m_5$. Here, $support(\{m_1, m_2\} \Rightarrow m_4) = 1$, $support(\{m_1, m_2\} \Rightarrow m_5) = 2$, $confidence(\{m_1, m_2\} \Rightarrow m_4) = 1/2$, and $confidence(\{m_1, m_2\} \Rightarrow m_5) = 2/2$. As $\{m_1, m_2\} \Rightarrow m_5$ having higher support, thus, the sorted impact candidate set F is $[m_5, m_4]$. However, two issues exist in the results: 1) *incomplete prediction*, where methods m_3 and m_6 are not included as they were not changed together with m_1 and m_2 in the query Q, and 2) *suboptimal ranking*, where the candidate method m_5 is falsely predicted and ranked higher than m_4.

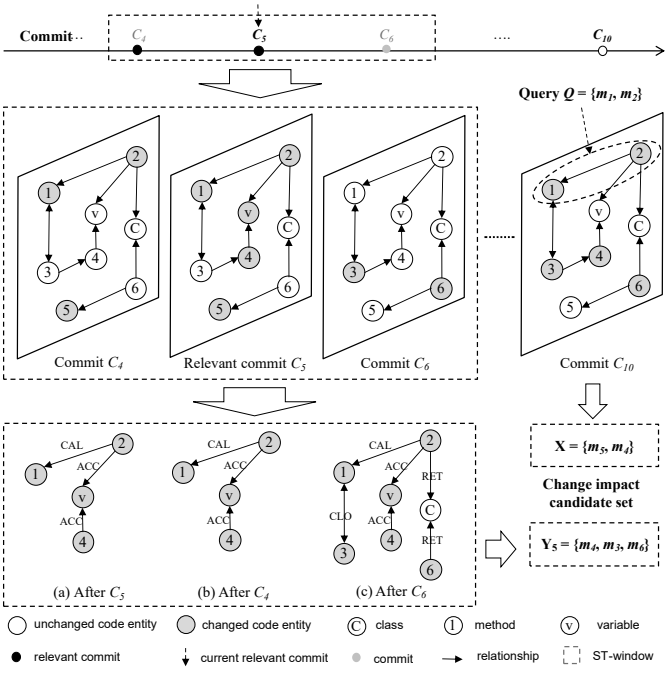


Fig. 1. Motivation Example

In this example, we find that the missing methods (m_3, m_6) were changed actually in the commit c_6 which is after c_5, and they were within a limited range (2-hops relationship) of the methods in the query Q. In contrast, the falsely reported method m_5 was far from the latter (3-hops relationship). This suggests that: 1) changes within a certain time and space should be considered to overcome incomplete prediction results; 2) temporal and spatial constraints could be utilized to eliminate less relevant prediction outcomes.

This insight motivates our approach IMPPRED, an approach to change impact prediction that employs change aggregation. By combining evolutionary coupling and various software relationships (such as structural dependencies and code clones) that reflect spatial and temporal change relationships among code entities, we can capture impacted code entities more comprehensively and improve prediction results, partially addressing limitations 1) and 2). Additionally, we introduce a new ranking strategy by analyzing the distance between impacted methods and methods in query Q (in terms of time and space) to improve prediction performance, partially addressing limitation 3). Section IV will provide details of IMPPRED.

IV. PROPOSED APPROACH

Fig. 2 provides an overview, where each ellipse represents a step, a rectangular box represents the intermediate results generated by each step, and a rounded rectangular box represents the input or output of the approach.

This paper focuses on predicting code changes at method level. We assume the availability of a complete code entity genealogy and SRGs dataset denoted as G. The initial set of changed methods is called an **initial query** Q. The commit set

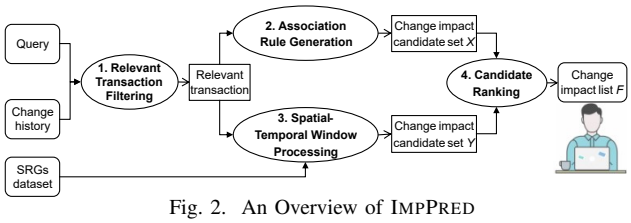


Fig. 2. An Overview of IMPPRED

of the project's history is denoted as $\mathcal{C}$, and for each commit $c \in \mathcal{C}$, it includes a list of changed code entities, denoted by CES_c, and the corresponding set of changed methods, called a **transaction**, represented by T_c. All transactions T_c form the transaction set of the project's change history, denoted by $\mathcal{H}$.

The IMPPRED involves four steps: 1) filtering the relevant transaction set based on the given query set (see Section IV-A), 2) generating association rules and obtaining the impacted candidate set (see Section IV-B), 3) considering the temporal and spatial change relationships to obtain the change impact candidate set (see Section IV-C), and 4) sorting the candidate sets to obtain the final change impact list (see Section IV-D).

A. Step 1: Relevant Transaction Filtering

This step aims to obtain the transaction set relevant to a given query Q. A transaction T is considered a **relevant transaction** if $T \cap Q \neq \emptyset$, and the corresponding commit is called as a **relevant commit**. We extend the scope of relevant transactions beyond conventional targeted association rule mining techniques, e.g., ROSE or TARMAQ, allowing us to capture more transactions related to Q for more comprehensive prediction results. We analyze each transaction $T \in \mathcal{H}$ to obtain the **relevant transaction set** with respect to Q, denoted as $\mathcal{H}_Q$, and the **relevant commit set**, denoted as $\mathcal{C}_Q$.

To conduct targeted association rule mining, we utilize TARMAQ's filtering rules and expand upon them to obtain the maximum relevant transaction set for Q, denoted as $\mathcal{H}_{MQ}$. A transaction $T \in \mathcal{H}_{MQ}$ if it satisfies the following two criteria: 1) $|T \cap Q| = k$, where k is the maximum intersection value between transaction T and query Q in the change history $\mathcal{H}$; and 2) $T - Q \neq \emptyset$. It is evident that $\mathcal{H}_{MQ} \subseteq \mathcal{H}_Q$. The $\mathcal{H}_{MQ}$ and $\mathcal{H}_Q$ will serve as inputs for Steps 2 and 3, respectively.

Example. In Fig. 1, we aim to predict the impacted methods for a given query $Q = \{m_1, m_2\}$ on target commit c_{10}. Based on Q, the relevant transaction set can be $\mathcal{H}_Q = \{T_4, T_5\}$. Both T_4 and T_5 satisfy the maximum relevant transaction definition, thus $\mathcal{H}_{MQ} = \{T_4, T_5\}$.

B. Step 2: Association Rule Generation

This step aims to generate relevant association rules for the query Q and obtain the corresponding change impact candidate set, denoted as X. To achieve this, each transaction T in the maximal relevant transaction set $\mathcal{H}_{MQ}$ is sequentially analyzed to construct the association rule $A \Rightarrow x$, where $A = T \cap Q$ represents the intersection of transaction T and query Q, and x denotes a single method from $B = T - Q \neq \emptyset$.

After analyzing all transactions in $\mathcal{H}_{MQ}$, the change impact candidate set X is generated by applying the association rules $A \Rightarrow x$ that were obtained. Here, $X = \{x\}$.

Example. In Fig. 1, we can obtain two association rules, i.e., $\{m_1, m_2\} \Rightarrow m_4$ and $\{m_1, m_2\} \Rightarrow m_5$. Therefore, the corresponding change impact candidate set is $X = \{m_4, m_5\}$.

C. Step 3: Spatial-Temporal Window Processing

The aim of this step is to analyze relevant commits, construct the **Spatial-Temporal Change Relationship Graphs** (ST-CRGs) by capturing changed code entities and their relationships within a limited time and space range, and obtain the corresponding change impact candidate set Y based on the ST-CRGs. We assume that code changes propagate within a specific time and space range. A **Spatial-Temporal window** (ST-window) is created around the relevant commit to define the time and space range, where the time range is defined by a *time window* (TW) and the space range is determined by the number of hops along the path of relationships, known as the *space window* (SW). An ST-CRG is generated for each relevant commit $c_i \in \mathcal{C}_Q$, leading to the corresponding change impact candidate set Y_i. By analyzing all relevant commits, we obtain the final change impact candidate set $Y = \{Y_i\}$.

The ST-CRG is a graph representation of code entities related to a relevant commit and their potentially changed software relationships. It comprises the code entities changed in a relevant commit (called *core entities*) and those that are potentially linked to the core entities based on their software relationships within an ST-window.

Definition 2: ST-CRG is a directed graph, G=(V, E), where V denotes the code entities and E denotes relationships between them. Each vertex in V is labeled based on its type (i.e., Class, Method, or Variable), and each edge in E is labeled based on its type of relationship (e.g., EXT, IMPL, CALL, etc). The vertices in V include three subsets of code entities: (1) the core entities changed in the relevant commit; (2) the changed non-core entities in the SRG of each commit in the time window, within k hops to the core entities (mapped from the relevant commit); and (3) the unchanged entities that are on the shortest paths between vertices in Subset (1) and Subset (2). Multiple edges are allowed between two code entities if there are more than one relationship between them.

For a given query Q, relevant commit c_i, relevant transaction T_i, and corresponding ST-window, we obtain the change impact candidate set Y_i following three steps:

1) *Determining the time window*: The *time window* is a period of time that covers a number of commits before and after the relevant commit. The definition of time window is different from the prior works [39]–[42], as we acknowledge that changes in the relevant commit may have been influenced by earlier commits, while also potentially affecting subsequent ones. Therefore, we analyze changes before and after the relevant commit to capture changes related to the core entities.

2) *Generating the ST-CRG*: The initial nodes of ST-CRG are established by adding the intersection $A = T_c \cap Q$. Code entities that change within the k-hop relationship range of the methods in A are added to ST-CRG along with the code entities that don't change but are on the k-hop paths. Simultaneously, the corresponding k-hop path relationships are

added. Afterward, other commits within the time window are sequentially analyzed, and the changed code entities within the k-hop relationship range of the methods in A (mapped from the methods in the relevant commit) and the unchanged code entities on the k-hop paths are added to ST-CRG. The corresponding k-hop path relationships are added simultaneously.

3) *Gaining the changed impact candidate set*: Based on the constructed ST-CRG, the set of changed methods within the k-hop relationships range of the query Q forms the change impact candidate set, denoted as Y_i.

Example. In Fig. 1, we explicate the process of creating an ST-window for a relevant commit c_5, acquiring the corresponding change impact candidate set Y_5. Here, the TW is set to 24 hours, and the SW is set to 2 hops. This TW covers three commits: c_4, c_5, c_6. We start by analyzing the relevant commit, then process other commits within the window to generate the ST-CRG and derive the impact candidate set.

For c_5, the intersection $A = T_5 \cap Q = \{m_1, m_2\}$. We add the methods m_1 and m_2 into the ST-CRG, and include method m_4 as it falls within the 2-hop range of methods in A_5. Conversely, we exclude method m_5 as it doesn't meet the criteria. We also include the variable v that lies on the path between certain methods, as shown in Fig. 1-(a). For c_4, as m_5 doesn't fall within the 2-hop range of m_1 and m_2, no node is added to the ST-CRG, as shown in Fig. 1-(b). For c_6, we include m_3 in the ST-CRG since it falls within the 1-hop range of m_1, and include m_6 as it falls within the 2-hop range of m_2. Moreover, nodes of class C are added on the path along with RET relationships, as shown in Fig. 1-(c).

Based on the generated ST-CRG, we obtain the changed methods within the 2-hop range of the query Q. The methods m_3, m_4, and m_6 meet the criteria, resulting in $Y_5 = \{m_3, m_4, m_6\}$ as the change impact candidate set for c_5.

Assuming that the ST-window generated via the relevant commit c_4 includes only commit c_4 and c_5, so $Y_4 = \{m_4\}$. Thus, $Y = Y_4 \cup Y_5 = \{m_3, m_4, m_6\}$.

D. Step 4: Candidate Ranking

This step aims to rank and generate the final change impact candidate list F by combining the candidate sets X and Y obtained from steps 2 and 3, respectively. Here, $F = X \cup Y$. A candidate method $f \in F$ can have multiple relationships with method q in Q, e.g., co-change or software relationships, which increase the possibility of f being impacted by a change. To rank the methods f, we employ a comprehensive ranking approach that sorts the candidate sets X and Y separately and then integrates them. The ranking order of candidate method f is determined the following formula:

$$rank(f) = \alpha * rankX_f + (1 - \alpha) * rankY_f \tag{1}$$

Here, $rankX_f$ and $rankY_f$ denote the positions of candidate method f in the sorted output of step 2 and step 3, respectively, where the ranking strategies are denoted as *RankX* and *RankY*.

The parameters α and $1 - \alpha$ represent weights assigned to the ranking strategies, reflecting their relative importance. These parameters satisfy the following two criteria: a) $\alpha \in [0, 1]$; b) $0 \leq rankX_f, rankY_f < |X \cup Y|$. If $f \in X$ and $f \notin Y$, then $rankY_f = |Y|$; if $f \in Y$ and $f \notin X$, then $rankX_f = |X|$. The first constraint modulates the importance of diverse ranking strategies, while the second considers the possibility of non-identical candidate impact sets X and Y, ensuring practicability. The subsequent sections introduce the two ranking strategies, *RankX* and *RankY*.

1) Strategy RankX: We use the *support* and *confidence* ranking strategy [11] to sort the impact candidate set X. The ranking values are determined by the support of the association rules, with higher values indicating higher rankings. In cases where two methods have equal support, confidence is used to determine their rankings, with higher confidence corresponding to higher rankings.

2) Strategy RankY: This strategy focuses on sorting the impact candidate set Y, considering temporal (change time intervals) and spatial (relationship hops) aspects. We assume that a candidate method f is more likely to be impacted by changes when it is closer to the query in terms of time intervals and relationship hops. We calculate the probability of a candidate method f being modified as follows.

$$p(f) = \beta * score_1 + (1 - \beta) * score_2 \tag{2}$$

Here, $score_1$ and $score_2$ denote the probability of the candidate method f being changed in the TW and SW, respectively. The weights β and 1 - β represent the importance of different factors. These parameters comply with two constraints: a) $\beta \in [0, 1]$; b) $score_1, score_2 \in (0, 1]$. Higher values of $score_1$ and $score_2$ indicate a higher probability $p(f)$ of the method f being changed, resulting in a higher ranking.

For $score_1$, we examine the time intervals between a candidate method f and the methods in Q, called the *change time intervals* (CTI). Shorter CTIs suggest a higher chance of impact on the candidate method, resulting in a larger $score_1$. To compute $score_1$, we first calculate CTI values for each relevant commit using a time window, then use the median value to represent the CTI between f and Q, finally, normalize the median CTI using an exponential function. Specifically, the CTI for f in relevant commit c_i is denoted as $CTI_i^{(f)}$, measured in hours, and calculated as follows:

$$CTI_i^{(f)} = \begin{cases} 0, \ if \ f \ co\text{-}changed \ with \ Q \cap T_i \\ min(|time(f) - time(c_i)|), \ otherwise \end{cases} \tag{3}$$

Here, $time$ represents the commit time. The set of CTI for f under all relevant commits is denoted as $\{CTI_i^{(f)}\}$. We calculate $score_1$ using the formula: $score_1^{(f)} = a^{\sigma * med(\{CTI_i^{(f)}\})}$, where a and σ are parameters of the exponential function. We set the $score_1$ to 1 when the CTI is 0 hours and 1/2 when the CTI is 12 hours. Thus, $a = E$ and $\sigma = -1/17.3$.

For $score_2$, we consider the spatial relationship between the candidate method f and the methods in query Q. We believe that closer proximity among code entities increases the likelihood of change propagation. The value of $score_2$ for f is calculated using the formula: $score_2^{(f)} = \frac{1}{k}$ $(k = 1, 2, ..., n)$,

where k represents the shortest distance (measured in hops) between the method f and the methods in the query Q.

Example. In Fig. 1, we set parameters α and β to 0.5 for the illustration. Two association rules, namely $\{m_1, m_2\} \Rightarrow m_4$ and $\{m_1, m_2\} \Rightarrow m_5$, can be generated through targeted association rule mining. Based on the $support$ and $confidence$ values (*RankX*), $F^{(X)}$ is sorted as $[m_5, m_4]$. However, $F^{(X)}$ may not be optimal (See Section III-B).

We proceed to further examine and sort the impact candidate set Y based on *RankY*. Following Step 3, $Y = Y_4 \cup Y_5 = \{m_3, m_4, m_6\}$. For m_4, it only changed in c_5, which co-changed with the methods in $Q \cap T_5 = \{m_1, m_2\}$. Thus, $CTI_4^{(m_4)} = CTI_5^{(m_4)} = 0$, and $score_1^{(m_4)} = 1$, while $score_2^{(m_4)} = 1/2$ due to its 2-hops relationship with m_2. The change probability for m_4 is $p(m_4) = 0.5*1+(1-0.5)*1/2 = 0.75$. Similarly for m_3, $CTI_4^{(m_3)} = CTI_5^{(m_3)} = 12$, resulting in $score_1^{(m_3)} = 1/2$, and $score_2^{(m_3)} = 1$ due to its 1-hop relationship with m_1. The change probability for m_3 is $p(m_3) = 0.5 * 1/2 + (1 - 0.5) * 1 = 0.75$. For m_6, $score_1^{(m_6)} = 1/2$, $score_2^{(m_6)} = 1/2$, and $p(m_6) = 0.5$. Therefore, $p(m_4) = p(m_3) > p(m_6)$, and the impact candidate set Y is sorted as $[m_4, m_3, m_6]$, denoted as $F^{(Y)}$.

Finally, the results of $F^{(X)}$ and $F^{(Y)}$ are combined for candidate ranking. Since the methods in X and Y are not entirely same, the missing methods are added to obtain $F^{(X)'} = [m_5, m_4, m_3, m_6]$ (where $rankX_{m_3} = rankX_{m_6} = 2$) and $F^{(Y)'} = [m_4, m_3, m_6, m_5]$ (where $rankY_{m_5} = 3$). Thus, $rank(m_5) = 0.5 * 0 + (1 - 0.5) * 3 = 1.5$. Similarly, $rank(m_4) = 0.5$, $rank(m_3) = 0.5$, and $rank(m_6) = 2$. The final ranking result is $F = [m_4, m_3, m_5, m_6]$.

Summary. IMPPRED reports more candidate results and mitigates the risks of missing potentially impacted entities. It also assigns higher priorities to the highly-correlated candidates. Our hybrid ranking strategy is designed to compensate for the shortcomings of a single strategy.

V. EVALUATION

We evaluate the performance of IMPPRED in predicting the impact of changes through simulations, comparing it with the state-of-the-art methods such as ROSE [10] and TARMAQ [11]. Moreover, we also examined the prediction results obtained using only the spatial-temporal window processing (called IMPPRED-ST). We assumed that the method of a transaction is related and partitioned the transaction into a query and an expected output. Our dataset contains 19,003 commits from six Java open-source projects. The implementation and data from our research are available online [43]. Our research questions (RQs) are as follows:

- **RQ1: Applicability**. Can considering change relationships among code entities enhance the applicability of change impact prediction methods?
- **RQ2: Effectiveness**. How effective is IMPPRED compared to the state-of-the-art methods?
- **RQ3: Parameter Sensitivity**. How do each parameter's sensitivity impact IMPPRED's effectiveness?

TABLE II
SUBJECT SYSTEMS

Systems	SRev	LRev	Timespan (days))	#Files in LRev	#Methods in LRev	#LOC in LRev
Cassandra	1	2,432	599	381	6,471	70,434
Maven	1	5,386	1,094	277	2,232	24,970
Jmeter	1	2,219	1,064	706	6,960	79,215
Commons-io	1	3,118	5,437	232	2,372	15,868
JfreeChart	1	3,176	2,578	658	9,380	102,239
Freecol	1	2,672	1,375	374	5,464	63,743

SRev = Starting Revision, LRev = Last Revision

- **RQ4: Strategy Contribution**. What are the contributions of different prediction strategies to IMPPRED's results?
- **RQ5: Efficiency**. How efficiently does IMPPRED predict change impact in terms of time cost?

To describe the evaluation, we first describe the subject systems used in the experiments, then detail the data preparation process, and finally, the evaluation process.

1) Subject Systems: We selected six high-quality Java open-source projects for our experimentation. They are: Cassandra [44], a distributed NoSQL database system; Maven [45], a project management and integration tool based on Java; Jmeter [46], a Java-based stress testing tool; Commons-io [47], a tool class library for handling IO; JfreeChart [48], an open chart drawing class library on the Java platform; and Freecol [49], a turn-based strategy game. The first four projects are all from the Apache Software Foundation, the latter two are well-known projects hosted on GitHub. They have a rich evolutionary history and are actively maintained. Additionally, these projects contain high-quality submission information, which makes them an appropriate foundation for validating our proposed method. We present detailed information about these software projects in Table II.

2) Data Preparation: We collected several types of data for each project, including the code entity genealogy, SRGs, and project transaction data (i.e., a set of changed methods).

Code Entity Genealogy. We established the code entity genealogy by mapping code entities across different versions to track their change status. File-level tracing is initially established using Git's built-in mechanism which tracks file movement and renaming. CLDiff [50], an AST-based code differencing tool, is then used to map code entities in adjacent commits by employing AST-based matching technology. Classes are mapped based on the mapped files, while variables are mapped based on their name and class name. Methods are mapped based on the AST matching results by tracking changes in method end line numbers across successive commits, which avoids inaccuracies resulting from changes to method signatures. Unique indices are assigned to all mapped entities, allowing tracking of all code entities in each commit. Deleted code entities are mapped to null entities, while new entities are assigned unique indices. To verify the accuracy of the code entity genealogy, two developers with rich Java development experiences investigated 200 randomly-selected cases, and confirmed over 95% accuracy of the genealogy.

Software Relationship Graphs. We first obtained code snapshots using Git. Then, we extended the dependency analysis tool *Depends* [51] and the code clone detection tool *SAGA*

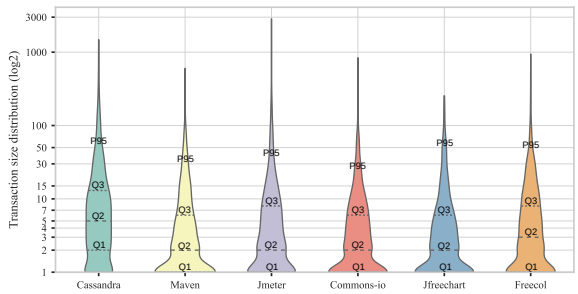


Fig. 3. Distribution of Transaction Sizes

[37] to extract 12 different types of relationships among code entities, including structural dependencies, containment, and clones. Finally, the resulting SRGs were stored in a Neo4J graph database for easy querying. This process was repeated for all code snapshots, resulting in the final SRGs dataset.

Transaction Data. We used CLDiff [50], a code difference analysis method, to analyze changes made in each commit and identify changed code entities. These changed entities were then aggregated at the class, member method, and member variable levels to obtain entity-level differences. The resulting list of changed code entities for each commit c was denoted as CES_c. By analyzing the list of changed code entities CES_c, we obtained the corresponding transaction T_c for each commit c. An empty transaction was considered if there were no changed methods in a commit. The set of all transactions T_c, denoted as $\mathcal{H}$, represents the project's change history.

3) Evaluation Process: Conceptually, a query Q represents a set of methods that a developer changed since the last synchronization with the VCS. To evaluate IMPPRED, we randomly generated queries to simulate a developer's mistake of forgetting to update a subset of T, starting from a transaction T [11]. The evaluation process involved four main steps: (a) selecting suitable transactions for query generation, (b) generating queries and their expected outputs, (c) executing the proposed method for each query to obtain the predicted result, and (d) computing relevant evaluation metrics. Further details on each step will be provided in the subsequent sections.

a) Transaction selection. We selected candidate transactions with sizes in the range of $|T| \in [2, M]$ to ensure that each transaction has at least two methods, which are necessary to construct the initial query and expected output. Larger transactions tend to contain irrelevant changes, making them unsuitable as query candidates [10], [52], [53]. The distribution of transaction sizes in six projects is shown in Fig. 3, where 75% of transactions contain 15 or fewer changed methods, and 95% of transactions contain 50 or fewer changed methods. Thus, we set the maximum transaction size to 50.

b) Query Generation. For each qualifying transaction T, we generated two non-empty subsets denoted as Q and $T-Q$, where Q represents the initial query, and $T-Q$ represents the expected output. For example, if a developer modifies five methods in a commit c, the corresponding transaction is $T = \{m_1, m_2, m_3, m_4, m_5\}$. If a random query $Q = \{m_1, m_2\}$ is generated, then $T-Q = \{m_3, m_4, m_5\}$ is the expected output.

We generated queries for all qualifying transactions $\mathcal{H}$ of each project and conducted experiments accordingly. The number of commits and queries are presented in the second and third columns of Table III.

TABLE III
EXPERIMENTAL SETTING

Systems	#Commits	#Queries (%)	#Commits /day	#Days/20 commit	TW (days)	SW (k-hop)
Cassandra	1,877	1,040 (55%)	3.13	6.38	6	2
Maven	2,540	929 (37%)	2.32	8.61	8	2
Jmeter	1,536	571 (37%)	1.44	13.85	14	2
Commons-io	1,553	409 (26%)	0.29	70.01	70	2
JfreeChart	1,798	505 (28%)	0.70	28.68	28	2
Freecol	2,265	1,211 (53%)	1.64	12.17	12	2

Commits = the commits that involving changes to code entities

c) Prediction. For each query, we applied four approaches, i.e., ROSE [10], TARMAQ [11], IMPPRED-ST, and IMPPRED, to generate a ranking list of impacted methods, respectively. The default ranking strategy of ROSE and TARMAQ was used, i.e, support and confidence. For IMPPRED, we experimented with different values of α, β, and the ST-window size. The default values of α and β were 0.8 and 0.5, respectively, and their impact will be discussed in Section VI-3 (RQ3).

We set the *time window* (TW) size based on each project's commit density, considering both the length of time and the number of commits. The default size of the TW (in days, D) is the average length of time per M (default value is 20) commits in the development history. The TW is set to $D/2$ days before and after the relevant commit, and the number of commits is limited to M. If more than M commits were included, we proportionally dropped out the commits from the start and end points to ensure that the time window contains no more than M commits. This was done to prevent an excessive number of changed code entities that might make change propagation analysis infeasible. For the *space window* (SW), we set the size based on the number of hops in the SRG (default value is 2). The settings for the TW and SW of each project are presented in the last two columns of Table III.

d) Evaluation. We evaluated the performance of IMPPRED using four metrics: *precision*, *recall*, *F1-score*, and *mean average precision* (MAP). Given a transaction T and its generated query Q, the expected result is $A = T - Q$. The ordered list of predicted impacted methods is denoted as F. The calculation formulas for the first three metrics are as follows, $precision = \frac{|A \cap F| \times 100}{|F|}$, $recall = \frac{|A \cap F| \times 100}{|A|}$, $F1\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. MAP is a comprehensive metric that considers the precision and order of predicted results, which is particularly useful in practical development scenarios. It is calculated as the mean of the Average Precision (AP) of all predicted results for a given query. The AP value can be calculated using the following formula: $AP(Q, F) = \sum_{n=1}^{|F|} precision(n) \cdot \Delta recall(n)$, where $precision(n)$ is the precision of the top-n candidate methods in the list F, and $\Delta recall(n)$ is the difference in recall improvement between the Top n and $n-1$ methods. An example calculation for AP is presented in Table IV.

VI. RESULTS

In this section, we present the results of research questions.

TABLE IV
EXAMPLE OF AVERAGE PRECISION CALCULATION

Initial Q: $\{m_2, m_4\}$, expected output: $\{m_1, m_5, m_3\}$

Rank(*n*)	Method	*Precision*(*n*)	Δ *Recall*(*n*)
1	m_1	1/1	1/3
2	m_2	1/2	0
3	m_3	2/3	1/3
4	m_4	2/4	0
5	m_5	3/5	1/3

$AP = 1/1 * 1/3 + 1/2 * 0 + 2/3 * 1/3 + 2/4 * 0 + 3/5 * 1/3 \approx 0.75$

TABLE V
APPLICABILITY RESULTS FOR EACH APPROACH

System	#Queries	ROSE	TARMAQ	IMPPRED-ST	IMPPRED
Cassandra	1,040	522 (50%)	930 (89%)	917 (88%)	940 (**90%**)
Maven	929	602 (65%)	872 (94%)	869 (94%)	883 (**95%**)
Jmeter	571	272 (48%)	438 (77%)	424 (74%)	443 (**78%**)
Commons-io	409	246 (60%)	350 (86%)	322 (79%)	355 (**87%**)
JfreeChart	505	128 (25%)	251 (50%)	246 (49%)	262 (**52%**)
Freecol	1,211	654 (54%)	1,132 (93%)	1107 (91%)	1143 (**94%**)

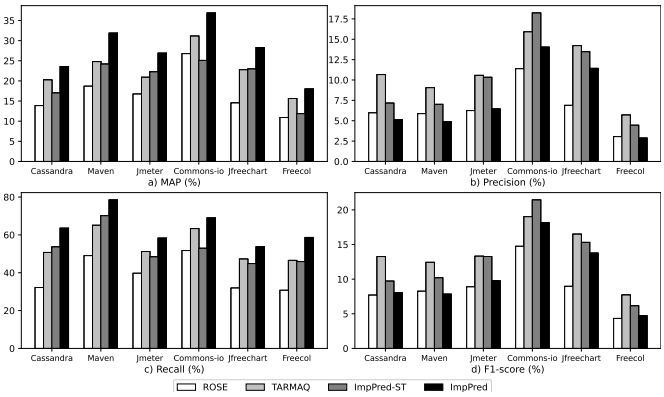
1) Applicability (RQ1): Table V presents the applicability results. IMPPRED achieves the highest proportion of predicted results across all projects, indicating its great applicability. IMPPRED outperforms TARMAQ by 2%, and outperforms ROSE by 64%. The applicability results of IMPPRED-ST is similar to TARMAQ with a slight decrease by 3%. This suggests that spatial-temporal window analysis is effective in obtaining the impact results for most queries, providing an alternative way for change impact prediction.

Answer to RQ1: That considering the change relationship among code entities can enhance the applicability of change impact prediction methods.

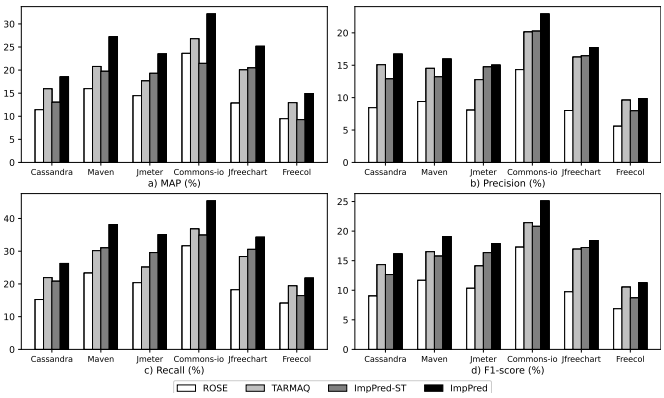
2) Effectiveness (RQ2): In this RQ, our evaluation was divided into two parts: 1) **Overall effectiveness**, which considers the complete set of change impact prediction results, and 2) **Top-5 effectiveness**, which assesses the practicality of IMPPRED by focusing on the Top-5 ranked results. The Top-5 prediction results are scrutinized since the number of methods that anticipate the expected result (real change) is typically less than 5 (See Fig. 3). Fig. 4 illustrates the efficacy evaluation outcomes for each method across different projects, while Fig. 4(a) and Fig. 4(b) display the results for the Overall effectiveness and Top-5 effectiveness, respectively.

Overall effectiveness: IMPPRED achieves the highest MAP and recall values among all projects. IMPPRED outperforms TARMAQ by 22% in MAP and 18% in recall, and outperforms ROSE by 63% in MAP and 62% in recall. However, IMPPRED is not superior in terms of precision and F1-score, possibly due to generating more candidate results which leads to more false positives despite improved recall. Nevertheless, IMPPRED utilizes a ranking strategy based on association rules and spatial-temporal window processing, allowing for higher rankings of candidate methods impacted by real changes. Previous works [11], [18], [33] have demonstrated the significance of ordered prediction results for developers.

Moreover, even considering only spatial-temporal window processing results (IMPPRED-ST), the method performs



(a) Overall Effectiveness



(b) Top-5 Effectiveness

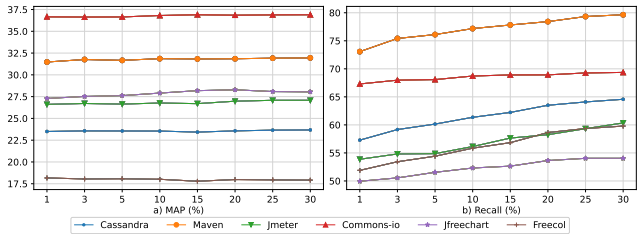
Fig. 4. Effectiveness Results for Each Approach

well. For example, in the *Commons-io* project, IMPPRED-ST demonstrates higher precision, and F1-score than TARMAQ. Importantly, IMPPRED-ST outperforms IMPPRED in terms of precision across all projects, indicating that spatial-temporal change relationships can capture impacted code entities more precisely and improve overall performance.

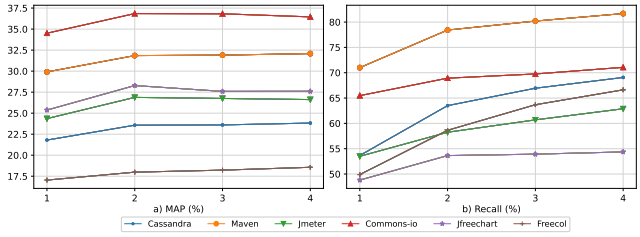
Top-5 effectiveness: IMPPRED shows the best MAP, precision, recall, and F1-score values in all projects. IMPPRED outperforms TARMAQ by 24% in MAP, 11% in precision, 24% in recall and 15% in F1-score, and outperforms ROSE by 61% in MAP, 82% in precision, 63% in recall and 66% in F1-score. Even considering only spatial-temporal window processing results (IMPPRED-ST), the method provides promising prediction results, with higher or similar recall in the *Cassandra*, *Maven*, *Jmeter*, and *Jfreechart* projects.

Answer to RQ2: IMPPRED outperforms the state-of-the-art methods with improvements up to 22% in MAP and 18% in recall. Its Top-5 effectiveness is also improved, with increases of 24% in MAP, 11% in precision, 24% in recall, and 15% in F1-score.

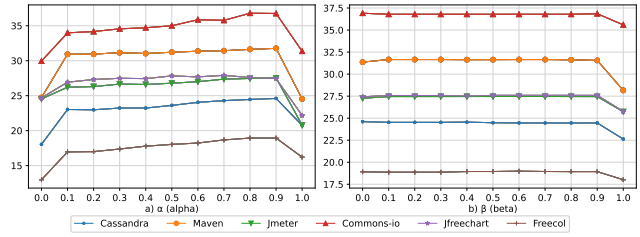
3) Parameter Sensitivity (RQ3): In the third step of IMPPRED, four thresholds can be configured: two window size thresholds, TW and SW, and two ranking strategy thresholds, α and β. The default configuration is 20 commits, 2-hop relationships, 0.8, and 0.5 (see Section V-3). To evaluate their sensitivity to the effectiveness of IMPPRED, we reconfigured one threshold and fixed the other three, and re-ran IMPPRED



(a) Time window (TW)



(b) Space window (SW)



(c) Two ranking strategy thresholds α and β

Fig. 5. Parameter Sensitivity Results

on our dataset. TW size is configured by the corresponding time span of eight different time windows with varying commit numbers. SW size is configured from 1 to 4 hops. α and β are configured from 0 to 1. After analyzing RQ2, IMPPRED exhibited superior performance in terms of MAP and recall, and thus we mainly compared the differences in these two metrics. Fig. 5 illustrates the impact of these thresholds on IMPPRED's MAP and recall, with the x-axis representing the threshold values and the y-axis representing the metrics.

Threshold TW: Overall, the recall of IMPPRED improves as the size of the TW increases, but the MAP varies among projects. Some projects, such as *Maven*, show an increase in MAP, while others, such as *Freecol*, experience a decrease. However, *Cassandra* shows relatively stable MAP values. When the maximum number of commits is restricted to 1, IMPPRED achieves a high MAP value but low recall rate. Increasing the TW size can improve recall but may also lead to more false positives, which can complicate candidate sorting. Nonetheless, our ranking approach remains effective, resulting in favorable outcomes while also improving recall. After considering recall and efficiency (See RQ5), we set the default TW size to a maximum of 20 commits, resulting in an average response time of around 2 seconds per query.

Threshold SW: As the hop count increases, the recall of IMPPRED continuously increases, but the MAP values vary among different projects. Specifically, in all projects, the MAP of IMPPRED improves considerably when the hop count is increased from 1 to 2. However, beyond 2 hops, the MAP values tend to either remain stable or decrease in some projects, while increasing in others. Based on the analysis of

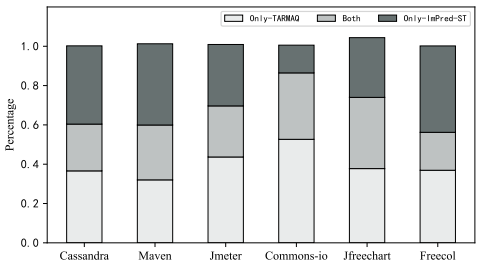


Fig. 6. Strategy Contribution in Different Projects

MAP values and running efficiency (RQ5), we conclude that a 2-hop is an optimal value for SW.

Threshold α and β: As α increases, the MAP of IMPPRED first increases, stabilizes, and then decreases. The highest MAP value is observed at approximately 0.8, with a significant improvement when α increases from 0 to 0.1 and a substantial decrease when α increases from 0.9 to 1.0. Combining support and confidence-based ranking strategy (*RankX*) with spatial-temporal window analysis (*RankX*) can further enhance IMPPRED's performance, achieving optimal sorting outcomes. Thus, we suggest setting α to 0.8. The relationship between β and MAP values of IMPPRED is complex, with some projects initially increasing, stabilizing, and ultimately decreasing, while others stabilize and then decrease as β increases. However, a notable decrease in MAP occurs when β increases from 0.9 to 1.0, highlighting the importance of spatial proximity. Thus, we set β to 0.5, balancing the impact of temporal and spatial aspects in candidate ranking.

> **Answer to RQ3**: The sensitivity of the configurable parameters to the effectiveness of IMPPRED is acceptable.

4) Strategy Contribution (RQ4): We assessed the effect of targeted association rule mining (S1) and spatial-temporal window processing (S2) on the prediction outcomes of IMPPRED. We compared the results obtained from S1 directly with those generated by TARMAQ. Our analysis categorized the IMPPRED predictions into three groups, based on the strategy that produced them: 1) *Only-TARMAQ*, 2) *Only*-IMPPRED-*ST*, and 3) *Both*, which represents the intersection of outcomes from both strategies. We calculated the mean percentage of each group, as shown in Fig. 6. Our results show that both strategies contributed significantly to the prediction outcomes, with an intersection ranging from approximately 19% to 36%.

> **Answer to RQ4**: Both targeted association rule mining and spatial-temporal window processing are crucial for IMPPRED's prediction outcomes, and their combination is reasonable.

5) Efficiency (RQ5): We conducted an experiment to assess whether IMPPRED meets the desired response time for method prediction in software development. The experiments were performed on a server equipped with a 2.4G Intel Xeon 4210R CPU and 64GB of memory, using a dataset comprising different projects with varying sizes ranging from 2,432 to 5,386 code commits, 2,232 to 9,380 methods, and 15,868 to 102,239 lines of code. The results, shown in Fig. 7, reveal

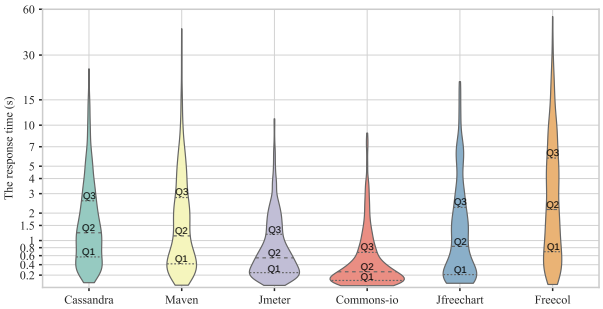


Fig. 7. The Distribution of Response Times

that 75% of queries were answered in under 3 seconds, and the median response time was approximately 2 seconds. Our findings suggest that IMPPRED meets the user's expectation for response time, as studies indicate that a typical response time is around 2 seconds [54].

Answer to RQ5: IMPPRED exhibits high operational efficiency overall and meets practical requirements.

VII. THREATS TO VALIDITY

Our approach largely relies on the accuracy of detecting static dependencies and code clones, as false detection can impact the validity of our results. To address this issue, we carefully selected and optimized our static analysis and clone detection tools, and manual verification on random samples revealed acceptable precision and recall levels.

Our experiments were conducted on a limited sample of six Java open-source projects, so the generalizability of our findings to other software systems developed in different languages or proprietary enterprise systems may be limited. However, we carefully selected our sample, and the projects represent a significant proportion of open-source projects, providing some degree of generalizability. We acknowledge that the number of revisions that we investigated may not be sufficient to establish our findings. Nonetheless, the selected systems have varying revision history lengths, as shown in Table II, indicating that our results are not biased by the revision history lengths of the subject systems.

VIII. RELATED WORK

We categorize change impact prediction methods into three types, including those based on association rule mining, code change pattern mining, and machine learning.

Association Rule Mining [10], [17] is a widely used that utilizes the evolutionary coupling relationship [1], [2], [29], [55] among code entities. For instance, Zimmermann et al. [10] developed a tool ROSE to suggest further code changes by the association rules. Rolfsnes et al. [11] proposed TARMAQ for mining evolutionary coupling and conducted change impact analysis in file level. Some researchers have conducted code change impact prediction at a finer-grained level, such as method or code snippet level [12], [13]. Although the association rule-based approach is useful for predicting the impact of code changes, it fails when changed code has never co-changed in the software's development history. Several techniques have been proposed to improve the performance, such as TAR [12], aggregated association rules [23], and adaptive recommendation [18]. However, these methods are limited by the number of association rules mined. Other researchers considered the combination of other relationships among codes, such as conceptual coupling [24], [25], [56], structural dependencies [3], [26], code clones [4], [5], [27], and requirement relationships [28]. These methods can provide more prediction results, but they have limited support for the ranking of candidate results.

Code Change Pattern Mining: Researchers have explored change impact prediction using code change patterns [7], [14], [57]–[59]. For example, Ying et al. [7] utilized frequent co-changed files as change patterns for prediction tasks. Wang et al. [58] proposed CMSuggester to suggest changes based on identifying $*CM \rightarrow AF$ change patterns where a modified method accesses a new member variable. Jiang et al. [14] extended CMSuggester's change pattern identification to $*CM \rightarrow AM$, increasing its applicability. Moreover, Koyuncu et al. [20] proposed automatic program repair through mining defect fix patterns. However, the limited availability of change patterns restricts change impact prediction to certain scenarios or patterns.

Machine Learning: Researchers have utilized machine learning techniques to extract features from changed code and related relationships from code change history for change impact prediction [15], [16], [60]. For example, Jiang et al. [15] identified six common code change patterns in JavaScript projects and applied machine learning to predict the impact of code changes based on the three most common patterns. However, their approach is limited to specific change patterns. Hu et al. [16] proposed a graph neural network-based model that integrates program dependency graph, code change information, and text difference information of clone code to predict consistent change requirements for cloned code without relying on clone code evolution history. Although these methods are effective for specific scenarios such as change patterns or code clones with a certain degree of similarity, their scope of applicability needs further enhancement.

IX. CONCLUSION

In this paper, we propose IMPPRED, a novel code change impact prediction method based on change aggregation. Our approach considers evolutionary coupling and spatial-temporal change relationships to improve the prediction completeness. Moreover, a candidate result ranking method is proposed to enhance the effectiveness of priority ranking. The performance of IMPPRED is evaluated on six high-quality Java open-source projects, involving evolution histories of totally 19,003 commits. Our evaluation indicates that IMPPRED outperforms the state-of-the-art methods with improvements up to 22% in MAP and 18% in recall. The Top-5 effectiveness is also improved, with increases by 24% in MAP, 11% in precision, 24% in recall, and 15% in F1-score. Furthermore, the results of sensitivity analysis, applicability analysis and efficiency analysis confirm IMPPRED's effectiveness and practicality.

REFERENCES

[1] H. C. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, November 16-19, 1998*. IEEE Computer Society, 1998, pp. 190–197. [Online]. Available: https://doi.org/10.1109/ICSM.1998.738508

[2] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *Proceedings of the 14th International Conference on Program Comprehension (ICPC 2006), 14-16 June 2006, Athens, Greece*. IEEE Computer Society, 2006, pp. 35–45.

[3] G. A. Oliva and M. A. Gerosa, "Experience report: How do structural dependencies influence change propagation? an empirical study," in *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*. IEEE Computer Society, 2015, pp. 250–260. [Online]. Available: https://doi.org/10.1109/ISSRE.2015.7381818

[4] M. Mondal, C. K. Roy, and K. A. Schneider, "Prediction and ranking of co-change candidates for clones," in *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*. ACM, 2014, pp. 32–41.

[5] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "Associating code clones with association rules for change impact analysis," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 93–103. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054846

[6] S. S. Yau, J. S. Collofello, and T. MacGregor, "Ripple effect analysis of software maintenance," in *The IEEE Computer Society's Second International Computer Software and Applications Conference, COMPSAC 1978, 13-16 November, 1978, Chicago, Illinois, USA*. IEEE, 1978, pp. 60–65. [Online]. Available: https://doi.org/10.1109/CMPSAC.1978.810308

[7] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *20th International Conference on Software Maintenance (ICSM 2004), 11-17 September 2004, Chicago, IL, USA*. IEEE Computer Society, 2004, pp. 284–293. [Online]. Available: https://doi.org/10.1109/ICSM.2004.1357812

[8] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. C. Chesley, "Chianti: a tool for change impact analysis of java programs," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, J. M. Vlissides and D. C. Schmidt, Eds. ACM, 2004, pp. 432–448. [Online]. Available: https://doi.org/10.1145/1028976.1029012

[9] Y. Wang, N. Meng, and H. Zhong, "An empirical study of multi-entity changes in real bug fixes," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018, pp. 287–298. [Online]. Available: https://doi.org/10.1109/ICSME.2018.00038

[10] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Software Eng.*, vol. 31, no. 6, pp. 429–445, 2005. [Online]. Available: https://doi.org/10.1109/TSE.2005.72

[11] T. Rolfsnes, S. Di Alesio, R. Behjati, L. Moonen, and D. W. Binkley, "Generalizing the analysis of evolutionary coupling for software change impact analysis," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 2016, pp. 201–212. [Online]. Available: https://doi.org/10.1109/SANER.2016.101

[12] M. A. Islam, M. M. Islam, M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "[research paper] detecting evolutionary coupling using transitive association rules," in *18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018, Madrid, Spain, September 23-24, 2018*. IEEE Computer Society, 2018, pp. 113–122. [Online]. Available: https://doi.org/10.1109/SCAM.2018.00020

[13] M. Mondal, C. K. Roy, B. Roy, and K. A. Schneider, "Fleccs: A technique for suggesting fragment-level similar co-change candidates," in *Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 2021, pp. 160–171.

[14] Z. Jiang, Y. Wang, H. Zhong, and N. Meng, "Automatic method change suggestion to complement multi-entity edits," *J. Syst. Softw.*, vol. 159, 2020. [Online]. Available: https://doi.org/10.1016/j.jss.2019.110441

[15] Z. Jiang, H. Zhong, and N. Meng, "Investigating and recommending co-changed entities for javascript programs," *J. Syst. Softw.*, vol. 180, p. 111027, 2021. [Online]. Available: https://doi.org/10.1016/j.jss.2021.111027

[16] B. Hu, Y. Wu, X. Peng, C. Sha, X. Wang, B. Fu, and W. Zhao, "Predicting change propagation between code clone instances by graph-based deep learning," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022*. ACM, 2022, pp. 425–436.

[17] T. Ball, J.-M. Kim, A. A. Porter, and H. P. Siy, "If your version control system could talk," in *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, vol. 11. Citeseer, 1997.

[18] L. Moonen, D. W. Binkley, and S. Pugh, "On adaptive change recommendation," *J. Syst. Softw.*, vol. 164, p. 110550, 2020. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110550

[19] J. Lee and Y. S. Hong, "Data-driven prediction of change propagation using dependency network," *Engineering Applications of Artificial Intelligence*, vol. 70, pp. 149–158, 2018.

[20] A. Koyuncu, K. Liu, T. F. Bissyandé, D. Kim, J. Klein, M. Monperrus, and Y. L. Traon, "Fixminer: Mining relevant fix patterns for automated program repair," *Empirical Software Engineering*, vol. 25, no. 3, pp. 1980–2024, 2020. [Online]. Available: https://doi.org/10.1007/s10664-019-09780-z

[21] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: how far are we?" in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '21, Virtual Event, Denmark, July 11-17, 2021*. ACM, 2021, pp. 427–438.

[22] Y. Qu, Q. Zheng, J. Chi, Y. Jin, A. He, D. Cui, H. Zhang, and T. Liu, "Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance," *IEEE Trans. Software Eng.*, vol. 47, no. 2, pp. 348–366, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2892959

[23] T. Rolfsnes, L. Moonen, S. Di Alesio, R. Behjati, and D. W. Binkley, "Aggregating association rules to improve change recommendation," *Empir. Softw. Eng.*, vol. 23, no. 2, pp. 987–1035, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9560-y

[24] H. H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard, "Blending conceptual and evolutionary couplings to support change impact analysis in source code," in *Proceedings of the 17th Working Conference on Reverse Engineering, WCRE 2010, 13-16 October 2010, Beverly, MA, USA*. IEEE Computer Society, 2010, pp. 119–128.

[25] H. H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, 2013.

[26] N. Ajienka and A. Capiluppi, "Understanding the interplay between the logical and structural coupling of software classes," *Journal of Systems and Software*, vol. 134, pp. 120–137, 2017.

[27] M. Nadim, M. Mondal, C. K. Roy, and K. A. Schneider, "Evaluating the performance of clone detection tools in detecting cloned co-change candidates," *Journal of Systems and Software*, vol. 187, p. 111229, 2022.

[28] D. Falessi, J. Roll, J. L. C. Guo, and J. Cleland-Huang, "Leveraging historical associations between requirements and source code to identify impacted classes," *IEEE Transactions on Software Engineering*, vol. 46, no. 4, pp. 420–441, 2020.

[29] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Proceedings of the 16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*. IEEE Computer Society, 2009, pp. 135–144.

[30] G. A. Oliva and M. A. Gerosa, "On the interplay between structural and logical dependencies in open-source software," in *Proceedings of the 25th Brazilian Symposium on Software Engineering, SBES 2011, Sao Paulo, Brazil, September 28-30, 2011*. IEEE Computer Society, 2011, pp. 144–153.

[31] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, P. Buneman and S. Jajodia, Eds. ACM Press, 1993, pp. 207–216. [Online]. Available: https://doi.org/10.1145/170035.170072

[32] R. Srikant, Q. Vu, and R. Agrawal, "Mining association rules with item constraints," in *Proceedings of the Third International Conference*

on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997, D. Heckerman, H. Mannila, and D. Pregibon, Eds. AAAI Press, 1997, pp. 67–73. [Online]. Available: http://www.aaai.org/Library/KDD/1997/kdd97-011.php

[33] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "Historank: History-based ranking of co-change candidates," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 240–250. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054869

[34] W. Pan, M. Hua, C. K. Chang, Z. Yang, and D. Kim, "Elementrank: Ranking java software classes and packages using a multilayer complex network-based approach," *IEEE Trans. Software Eng.*, vol. 47, no. 10, pp. 2272–2295, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2946357

[35] D. Cui, T. Liu, Y. Cai, Q. Zheng, Q. Feng, W. Jin, J. Guo, and Y. Qu, "Investigating the impact of multiple dependency structures on software defects," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 584–595. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00069

[36] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "An empirical study on bug propagation through code cloning," *J. Syst. Softw.*, vol. 158, 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.110407

[37] G. Li, Y. Wu, C. K. Roy, J. Sun, X. Peng, N. Zhan, B. Hu, and J. Ma, "SAGA: efficient and large-scale detection of near-miss clones with GPU acceleration," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 272–283. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054832

[38] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. A. Williams, Eds. ACM, 2016, pp. 1157–1168. [Online]. Available: https://doi.org/10.1145/2884781.2884877

[39] K. Herzig and A. Zeller, "Mining cause-effect-chains from version histories," in *IEEE 22nd International Symposium on Software Reliability Engineering, ISSRE 2011, Hiroshima, Japan, November 29 - December 2, 2011*, T. Dohi and B. Cukic, Eds. IEEE Computer Society, 2011, pp. 60–69. [Online]. Available: https://doi.org/10.1109/ISSRE.2011.16

[40] F. Jaafar, Y. Guéhéneuc, S. Hamel, and G. Antoniol, "Detecting asynchrony and dephase change patterns by mining software repositories," *J. Softw. Evol. Process.*, vol. 26, no. 1, pp. 77–106, 2014. [Online]. Available: https://doi.org/10.1002/smr.1635

[41] F. Jaafar, Y.-G. Gueheneuc, S. Hamel, and G. Antoniol, "An exploratory study of macro co-changes," in *Proceedings of the 18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011*. IEEE Computer Society, 2011, pp. 325–334.

[42] Q. Feng, Y. Cai, R. Kazman, D. Cui, T. Liu, and H. Fang, "Active hotspot: An issue-oriented model to monitor software evolution and degradation," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 986–997. [Online]. Available: https://doi.org/10.1109/ASE.2019.00095

[43] "Implementation and data," https://doi.org/10.5281/zenodo.7885439, 2023.

[44] "Cassandra," https://github.com/apache/cassandra, 2023.

[45] "Maven," https://github.com/apache/maven, 2023.

[46] "Jmeter," https://github.com/apache/jmeter, 2023.

[47] "Commons-io," https://github.com/apache/commons-io, 2023.

[48] "Jfreechart," https://github.com/jfree/jfreechart, 2023.

[49] "Freecol," https://github.com/FreeCol/freecol, 2023.

[50] K. Huang, B. Chen, X. Peng, D. Zhou, Y. Wang, Y. Liu, and W. Zhao, "Cldiff: generating concise linked code differences," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 679–690. [Online]. Available: https://doi.org/10.1145/3238147.3238219

[51] "Depends," https://github.com/multilang-depends/depends, 2023.

[52] L. L. Silva, M. T. Valente, M. de Almeida Maia, and N. Anquetil, "Developers' perception of co-change patterns: An empirical study," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, R. Koschke, J. Krinke, and M. P. Robillard, Eds. IEEE Computer Society, 2015, pp. 21–30. [Online]. Available: https://doi.org/10.1109/ICSM.2015.7332448

[53] L. Moonen, S. Di Alesio, D. W. Binkley, and T. Rolfsnes, "Practical guidelines for change recommendation using association rule mining," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 732–743. [Online]. Available: https://doi.org/10.1145/2970276.2970327

[54] F. F. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[55] D. Zhou, Y. Wu, L. Xiao, Y. Cai, X. Peng, J. Fan, L. Huang, and H. Chen, "Understanding evolutionary coupling by fine-grained co-change relationship analysis," in *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, Y. Guéhéneuc, F. Khomh, and F. Sarro, Eds. IEEE / ACM, 2019, pp. 271–282. [Online]. Available: https://doi.org/10.1109/ICPC.2019.00046

[56] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "Id-correspondence: a measure for detecting evolutionary coupling," *Empirical Software Engineering*, vol. 26, no. 1, p. 5, 2021.

[57] H. H. Kagdi, S. Yusuf, and J. I. Maletic, "Mining sequences of changed-files from version histories," in *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006*. ACM, 2006, pp. 47–53.

[58] Y. Wang, N. Meng, and H. Zhong, "Cmsuggester: Method change suggestion to complement multi-entity edits," in *Software Analysis, Testing, and Evolution - 8th International Conference, SATE 2018, Shenzhen, Guangdong, China, November 23-24, 2018, Proceedings*, ser. Lecture Notes in Computer Science, L. Bu and Y. Xiong, Eds., vol. 11293. Springer, 2018, pp. 137–153. [Online]. Available: https://doi.org/10.1007/978-3-030-04272-1_9

[59] A. Agrawal and R. K. Singh, "Predicting co-change probability in software applications using historical metadata," *IET Software*, vol. 14, no. 7, pp. 739–747, 2020.

[60] T. Rolfsnes, L. Moonen, and D. W. Binkley, "Predicting relevance of change recommendations," in *Proceedings of the 32th IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*. IEEE Computer Society, 2017, pp. 694–705.