ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

Mining Software Repositories on Integrating Orthogonal Information Retrieval Method Improving Automated Ir-Based Traceability Processes

[1]J. Arun Paul and [2]Prof. Dr. S. Suresh Kumar

[1]*Research Scholar, Sathyabama University, Chennai, Tamil Nadu, India*
[2]*Principal, Vivekanandha College of Technology for Women, Namakkal, Tamil Nadu, India*

ARTICLE INFO

Article history:
Received 12 March 2015
Accepted 28 April 2015
Available online 5 May 2015

Keywords:
Requirement Traceability, Source Code, Trust, Software Artifacts

ABSTRACT

Background: This paper methodology is to improve traceability links among software artifacts through the analysis of software system's form antiquity is obtainable. **Objective:** A heuristic-based methodology that uses consecutive pattern mining is functional to require in software repositories for recognition extremely frequent co-changing sets of artifacts (e.g., source code and certification). If diverse forms of files are organized together with high frequency then there is a high possibility that they consume a traceability link among them. **Results:** The method is estimated on a number of descriptions of the open basis structure KDE. **Conclusion:** As an authentication step, the discovered links are used to predict related changes in the innovative versions of the identical system. The results show exceedingly precision predictions of convinced types of traceability links.

© 2015 AENSI Publisher All rights reserved.

To Cite This Article: J. Arun Paul and Prof. Dr. S. Suresh Kumar., Mining Software Repositories on Integrating Orthogonal Information Retrieval Method Improving Automated IR-Based Traceability Processes. ***Aust. J. Basic & Appl. Sci.,*** *9(11): 334-342, 2015*

INTRODUCTION

Traceability link retrieval has been a theme of analysis for many years within the software engineering civic (Gethers, M., 2011). Consuming categorical recognized traceability links among various artifacts (e.g., source code and documentation) is vibrant for a variability of software maintenance tasks with impact analysis, program comprehension, and requirements declaration of high quality systems. It is particularly useful to support the understanding of the source code in

The case of links between source code, design, and documentation requirements. These links help to explain why a job or a certain category found in the program. Many technologies to assist in the recovery / Presented found in existing software systems to track links (Capobianco, G., 2013). However, the problem has shown that it is very difficult for any one approach is, in itself, completely accurate or successful. Many approaches suffer from the production of many false positives, indicating a link when there must be nothing (De Lucia, A., 2012). Approach to recover / discover tracking links analysis usually takes one (ie the current version) of the software system to infer links between two or more artifacts. The method presented here to detect trace links differs in that it examines the multiple versions (ie the date of the change) of a program to recover artifacts tracing the links. In (Ghafari, M., 2014) and put the hypothesis that if objects of different types (for example, src.cpp and help.doc) is the co-change with high frequency through multiple versions, then these artifacts may have trace link between them.

This history-based approach offers some distinct advantages over single version approaches, namely:

- Derived trace of actual changes in the business and technical ties, rather than estimates based on the various dependencies between structural and semantic analysis in one shot of the system.

To achieve this, we mine version of history found in the software that is retained by the version control such as vandalism or tools CVS repositories. Specifically, we use data mining and sequential pattern mining technique (Cuddeback, D., 2011), to identify and analyze groups of files that are committed together. This technique produces often participated in the changed files, known as the patterns of change. Change patterns that include files representing different types of artifacts are considered related via a traceability link.

Another approach, mining itemset (De Lucia, A., 2013), has also been used in the history of mining copy. Itemset mining produces patterns of change

Corresponding Author: J. Arun Paul, Research Scholar, Sathyabama University, Chennai, Tamil Nadu, India.
E-mail: arulpaul.me@gmail.com

that are unordered collections of co-changing objects; while mining sequential pattern is produced (in fact partially ordered) lists common variable objects. This means that the order in which objects have been changed (or obligation) and maintained. And so can be used to request information to infer the direction of tracking links.

Our approach is to assess the KDE open source software system (K Desktop Environment). The results shows that our approach is able to detect trace the links between different types of software artifacts (eg, source code, and change files, records, and user documentation, and build files) with high accuracy. It can be can easily apply this approach, along with the recovery of other connection methods, to produce a more complete traceability software system image.

Literature Survey:

There are two separate areas of research that are directly related to our work, which is mining software repositories (MSR) and recovery tracking link.

Kagdi *et al*. contemporary a survey of MSR tactics in the context of software evolution. Our work closely narrates to the works by Zimmermann *et al*. and Canfora *et al*. though with significant discrepancies. Zimmermann *et al*. essentially focused on exposure source-code to source-code change addictions using itemset mining. They deliberated only the source code entities committed together in a single change-set (approximated via sliding-window technique). Our focus is on uncovering the traceability links between source code and other types of artifacts (note that we also uncover source-code-to-source-code change dependencies). We also consider files committed over a sequence of change-sets and not just in a single change set. We use sequential-pattern mining to uncover the ordering information of committed files. Yang *et al* used a related method as Zimmermann *et al*. for detecting files that frequently variation together.

Canfora *et al*. work is established on the textual correspondence of diverse bug reports (and commit messages) in the change history. An information retrieval technique is used to guide the changed files in the CVS repositories with the textual explanation of past bug reports in the Bugzilla repository and the CVS commit communications. Our work is centered on a communal set of files that is different multiple times in the change history. As such, their work is reliant on the "quality" of the textual description. Moreover, they are only capable to find traceability among bugs/features and source code files for those bugs/features cross the threshold in the bug-tracking system. Our method can function without this info.

Spanoudakis and Zisman directed a comprehensive study of several methods for link recovery that activity such things as information retrieval, test-cases, and design patterns. Marcus and Maletic cast-off Latent Semantic Indexing (LSI) to recover links, with improved precision, from documentation to source code on the similar set of case readings done by Antoniol *et al*. using vector space IR models. These methodologies do not deliberate, or depend on, various versions of a software structure to concept the links. In one of the rare studies that reviewed two varieties at a time, Antoniol *et al*. form traceability links among software releases of an object oriented system to regulate contradictions.

Egyed practice program run-time data using test cases to expose traceability information between software artifacts. Spanoudakis and Zisman use heuristics for involuntary generation of traceability links among requirements and the UML object classical as well as between altered parts of a requirements document. Murphy *et al*. present software reflexion simulations for classifying links among high-level models and source code.

Proposed Method:

Our research attention is in recognition traceability among source code and other artifacts. This contains user documents (e.g., *HTML, XML/docbook, LaTeX and Doxygen*), form organization documents (automake, cmake, and make file), How To directors (e.g., *FAQs*), issue and dissemination documents (e.g., *ChangeLogs, whatsNew, README,* and *INSTALL* guides), development observing documents (*TODO* and *STATUS*), and extensible devices *(e.g., Python, Ruby,* and *Pearl* bindings for an *API*). These artifacts can be considered informalisms software (Bavota, G., 2014) Our approach is to analyze groups of files that often co-occur in groups change through frequent pattern mining technique application (ie, mining sequential pattern). We refer to such a group of files style change. Then analyze these patterns of change to detect patterns that contain code source files and other file types. We refer to this type style trace. Our hypothesis is that if the same set of files, of different types, and participated in the change with high frequency then there is a possible link between the trace. For this to be a sound hypothesis, the basic requirement is to study whether different types of files are usually changed together. Our study shows the six open source systems (Ali, N., 2014) that between 28% and 62% of the groups change (ie checking a set of files to the software repository together to commit a single process) containing two or more types of artifacts. These systems are covered in a number of application areas, and sizes, written primarily in C, C ++, and Java. Apache httpd is a web server, jEdit is the editor, the GCC is a translator, KOffice is a set of desktop applications, kdelibs is the core library for KDE (K Desktop Environment), and Python is a programming language. We believe such a change groups are a valuable source for detecting trace links (Qusef, A., 2010). The use of this source, an approach that

Australian Journal of Basic and Applied Sciences, 9(11) May 2015, Pages: 334-342

detects a specific set of files with tracking them can be placed links. First, we describe how the change is stored and groups represented in the software repositories to help facilitate the following discussion of our mining approach for tracking links.

Change-Sets in Software Repositories:

Storage depots metadata source code, such as user ID, time stamp, and commit comments in addition to the artifacts and source code differences across versions (Panichella, A., 2013). This definition explains why, who, and when the dimensions change the source code. Modern source control systems, such as vandalism, assemble and maintain many of the changes in multiple files into a single change as carried out by the perpetrator. Linked to the task version number and metadata to change control and recorded as an entry-level record.

Let's look at the problem with some assumption. For example, in the banking project in version 1 Developer mainly concentrated on the "unit balance inquiry" and let it complete.

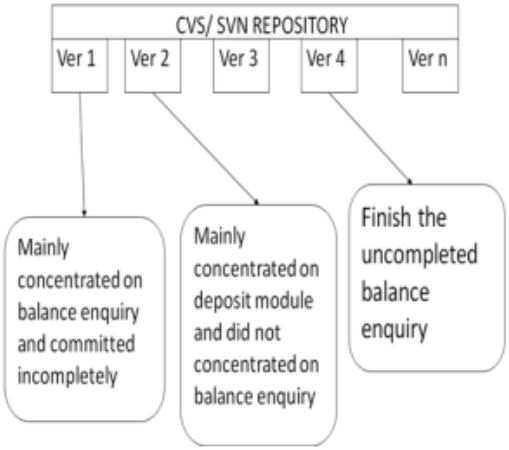

Fig. 1: Problem in Tracing Requirements.

In version 2, the developer mainly concentrated on the "unit trust" and did not focus on achieving balance. In version 4, and the completion of the "unity of balance," which was lacking in version 1. Similarly, in some version of the CVS / SVN developer warehouse some of the requirements taken and kept and which are completed but in the future the developer to complete those requirements have left incomplete in version later. Here the solution is to follow a certain cvs/svn contains the source code for the completion requirement.

Corresponds to the log entry process and one committed. This information can be easily obtained in an XML format using the client OS log command line. The entries include sabotage and record the dimensions of the author and history trails participate in-group change. In this case, the implementation of changes in khtml_part.cpp files and loader.h together by the developer Kling on a date / time 2005-07-25T17: 46: 20.434104Z. Is set revision number 438 663 for the entire change group (not all files that are changed as in the case with some version control systems like CVS). In addition, it is also recording a text message describing the change entered by the developer. Note that the order in which files appear in the log entry is not necessarily the order in which they were changed. It is clear that one record alone is not enough to give a chronological order in which files have been changed entry. However, there is a change between the time the system sets. Change groups, with increasing revision numbers occur after those with less revision numbers. Therefore, we can take advantage of the change order groups to determine the order of the changes between the different files. In the rest of the paper we use the term change groups to log entries in Subversion repositories. Let us now describe the terms of relevant data mining techniques and frequent pattern mining.

Enter data into algorithms drilling pattern is repeated in the form of a deal. Treatment refers to a group of elements that share a common ownership or occur in the same event (for example, check the customer baskets or items out together in the market basket analysis). The number of transactions in which the pattern is happening as it is known to support. The support of the pattern is the number of groups that seem. Even if the support of a pattern of at least the minimum specified by the user support after that is a recurring pattern in the data set in mind. Mining sequential pattern produces an ordered list of files to partially patterns and as such we call these patterns is.

Sequential pattern mining takes a set of sequences, and find each successive which occur frequently (ie, patterns of order) that have at least the minimum specified by the user support (Gethers, M., 2012). Here, the transaction is in the form of a sequence of items. And apply sequential pattern mining techniques usually to data sets with temporal

information or other demand. For example, in the case of market basket analysis with additional timestamp information, and sequential patterns such as customers who bought the camera is likely to buy additional memory in the next month as well.

Open Source and Traceability:

Scacchi *et al*. Note that the elicitation and analysis requirements and specifications of the open-source system is very different from the traditional approach (for example, the use of mathematical logic, and plans and descriptive models and design UML) in software engineering. Usually implicit requirements through the speech of the participants in the project, and after assurances implementation. Different types of non-official sources (described as software in formalities) constitute collective requirements and documenting open source project. This includes software, and communications warehouses, and evidence of Comedy, and documents the traditional system (for example, the man pages). One particular type of requirements that are a common feature in many open source projects is the ability to support the extension mechanisms with different programming languages and architecture (for example, a snake-binding libraries KDE). Due to the nature of the collaborative development of open, distributed, and includes software repositories main site for the project works of art along with the primary means of coordination and archives. And can see the bug / issue tracking and e-mail messages as a source of requirements and requests for corrective maintenance of the system open source repositories. Can be seen as a source control repositories sources of artifacts implementation. Few efforts have been made to the conclusion then use trace links between artifacts in repositories bug and antiques source code by mining software repositories (MSR). Canfora *et al*. The use of descriptions of illness and CVS commit messages for the purpose of forecasting change. Approach provides a set of files (for line-level granularity) that are likely to change because of the textual description of the existence of a new fault (or feature). Is used to retrieve information to files that have been changed in the CVS repositories with a text description of the last bug reports in Bugzilla repository and CVS commit messages index method. Associated with a bug report to the CVS commit (ie a set of files that have been changed), based on clear faulty ID found (a practice common in the open source development) in the commission of that message (for example, bug ID 30,000). Sliwerski *et al*. used a combination of information in a log file CVS (commits) and Bugzilla to study changes spur reform. Changes which stimulates reform changes are introduced new changes to fix a problem reported earlier.

```
<?xml version="1.0" encoding="utf-8"?>
<log>
  <log entry revision="438663">
    <author>kling</author>
    <date>2005-07-25T17:46:20.434104Z</date>
    <paths>
      <path action="M">khtml_part.cpp</path>
      <path action="M">loader.h</path>
    </paths>
    <msg>
       Do pixmap notifications when
       running ad filters.
    </msg>
  </log entry>
</log>
```

Fig. 2: A Snippet of kdelibs Subversion Log.

Required to process the order to give a proposal for the developer to automatically find that contains a copy of the source code to meet their requirements. Since the similarity accuracy less text, occupies the place is used in the second, third and options to meet the need to trace the development of the condition. The implementation of the ranking process by maintaining the requirement by maintaining a link tracking condition (RTL) (Ali, N., 2011). Track condition link is a set containing the clause, and the source code and the value of its similarity elements. Here to explain, version1 presence tracking condition k {R1, S1, SV = 6% LINK} and {version 2 R1, S1, SV = 9%} and version 3 {R1, S1, SV = 5%}. While the comparison between the three versions of Version 1 ranked in second place, version 2 is in the first position and version 3 third place classified.

This arrangement gives the proposal a certain condition trace to find the full source code version expressive or completed to the formulation of the final product program and also in the development of software engineering phase.

He described the text in Bugzilla along with heuristics to determine the Delta CVS associated change that fixes the flaw being the use of regular expression matching to commit messages. Cubranic *et al*. describes a tool, a Hipikat, to help new developers (not necessarily novice) on the project, in the performance of the current task (s). Recommends Hipikat artifacts from the project, which could

complicate the task at hand link memory. Developer may request for the relevant clearly artifacts in the form of an explicit query, or a tool that can do it automatically, based on the current context (for example, based on the currently open document (s) in the developer workspace). In short, focused approach based on MSR detect trace links between applications in the bug tracking systems and source code. While these efforts are important, but they do not cover only part of a wide range of documents found in the open source development. And sustainable success of the open source project, from both the development and the end points of view to use, depends largely on the extent of their success in preserving these documents. For example, the application can fail often to assemble or with very little installation instructions have the effect of diminishing the user base. It is important that these documents remain in alignment with the current state of the source code. Therefore, the trace of them is desirable interest and value (Corley, C.S., 2011). These documents represent as well as requests to track a major step towards a complete picture of the trace to the source code in the context of the Open Source Development Bug systems.

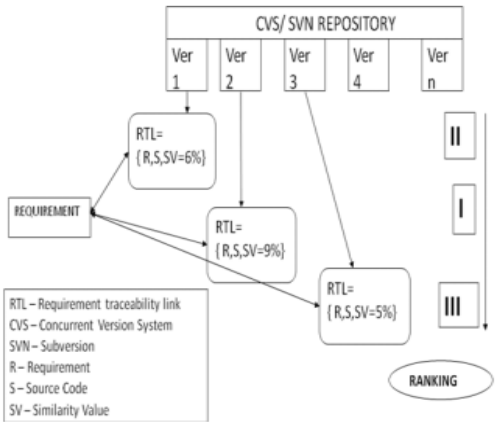

Fig. 3: Ranking Process of RTL.

Methodology:

We have developed a sequential pattern mining tool, two sqminer be based on sequential pattern Discovery algorithm (SPADE), which uses census efficiency of the order on the basis of the common prefix and successive division of space research using equivalence classes patterns. In addition, it uses the introduction of vertical transactions formula (ie a set of transactions for each file against a set of transactions that consist of files) to achieve efficiency. To help trim the number of candidate patterns produced by mining techniques, and patterns are eliminated with redundant information. A pattern that is repeated means that all possible patterns formed from subsets of files are also frequent. The support of the pattern is always less than or equal to the subtypes. Pruning common mechanism used in the frequent pattern mining is to eliminate all subtypes that have the same support from the corresponding larger pattern. These patterns are used only a subset with other larger patterns and not in isolation of its own (Ali, N., 2012). Thus, they give redundant information that may be of very little meaning. As a result, only a disjoint patterns (ie patterns with any shared files) are kept to accommodate all subsets of patterns with the same or higher support. It is known that these patterns for closed patterns. Our tool produces only closed patterns.

Frequent exploration algorithms pattern usually report the support of the style but not the transactions in which it occurs. Our tool transactions that were found on the pattern records. To detect each of the patterns of change unordered and ordered, we use the same basic mining algorithm. Can also be sqminer tool can be used for mining frequent itemset. In this case the formation transactions with any information to order items. Includes configuration parameters of sqminer support, the maximum number of elements in the pattern and sequence of mining (association) rules and production in each of the flat file format and XML. For more details about the XML output format of patterns and rules ordered, we refer to (Borg, M., 2013).

This is based on a heuristic assembly on the grounds that the change sets that were committed during the period of the intervals are related, and change the groups committed outside this interval and irrelevant. All change groups committed in a certain period of time in one group are placed. The number of groups is equal to the number of unique time periods of its kind over committed in groups change. This covers heuristic change-related groups that are committed by different developers, but during the same time period.

For example, have been found on the pattern {khtml_part.h}→{change log} of mining groups change in KDE Subversion repository (under kdelibs

/ KHTML /) committed in the period between May 2005 and December 2005. In this case, the team was formed Groups of change that were committed in one calendar day. Each day of the five days, and was khtml_part.h file in-group change committed by the committed change log file status change. This is the tracking pattern shows that the changes are documented after changing the interface file.

$$\{kdeedu / kalzium / src / kalzium.cpp,$$
$$kdeedu / kalzium / src / pse.cpp\} \rightarrow$$
$$\{kdesdk / doc / scripts / kdesvn - build / index.docbook\}$$

(Ali, N., 2013) Pattern is a pattern is another example of the change that occurs in groups that were committed in each of five different days. This pattern indicates that the documents are updated after making changes to the source code. However, it cannot determine the order in which files have been changed source code (any style is in part).

Experimental Result:

Applications in KDE represent a wide range of areas, programming languages, and size, and developers. Assessment methodology is the first part of the mine, the date of issuance of the tracking patterns. We call this the training set. We are next to mine later part of the date of issue (called the evaluation group) and see if the results generated from the training group can accurately the changes that occur in the evaluation of a range prediction. We have seen a change in the groups that were committed in the twelve-month period in KDE repository from 01/05/2005 to 30/04/2006. KDE migrated from CVS to Subversion in May 2005, and this was the main reason we have to choose this time frame. Set aside 2/3 of this version of history to train a group: an eight-month period starting from the date on 01/05/2005 and ends on 12/26/2005. This training kit contains 14 939 change sets consisting of 13037 files. Devoted the rest of the evaluation group: four months from the date of start and end on 2005-12-27 2006-04-30 (Ali, N., 2012). This set-evaluation contains 9008 revisions (no change groups), and consists of 9070 files. Change is the only groups that consist of ten files or less.

This change avoids groups such as updating the license information on each file or merge and copy performance. First we need to extract the relevant information from the KDE repository. Direct approach to extract the log entries of subversion repository is the use of feed record of leadership in the client version of the operating inventory. This approach is not feasible for in-use cases that need only records stored in the warehouses and the program is not committed to the contents of the documents. We change extractor tool used pysvn (module for Python vandalism) to extract the change groups, without the use of a copy of the inventory. Extractor change takes repository URL, start date and the date of the end of history, and extracting the change sets of records repository for a specific period. Then assemble the extracted change groups from the warehouse to the sequence according to infer changes assembly tool set (another Python script). We choose a day calendar as the time interval for heuristic interval. Is set and the perpetrator of that contributed to a change on the set of the perpetrator of the heuristic. A number of groups and a number of groups involved in the change of these groups for the three heuristics shown in Table 1.

Table 1: Groups designed from the change-sets mined from the KDE repository by the altered heuristics.

Heuristics	Groups
Day	237
Committer	330
Committer Day	4,884

Table 2: Change patterns (CP) and traceability patterns (TP) uncovered from the KDE repository.

Heuristics	CP	TP	TP/CP%
Day	5,839	1,851	37.10
Committer	718	54	7.52
Committer Day	2,372	277	11.68

Uncovering Traceability Patterns:

The groups assembled by the tool group deviations are fed to the sequential-pattern mining tool sqminer. Mining frequent methodical patterns with sqminer produces a set of barred change patterns. We configured sqminer to mine patterns with a minimum of five for all the support we have specific heuristics. Table 2 shows the patterns of change and the exposed trace patterns revealed by inference. One trace pattern consists of at least the source code file and at least one other type of file. C ++ is a KDE primary programming language. With extensions {.h, .cpp, .cc, .c, .hxx, .cxx} considered that contain the source code, while those that help files are not the source code as other works of art (e.g., with extensions .docbook, .xml, and .html, and ChangeLog) (Jaafar, F., 2014). We do not restrict other works of art for a given set of types. Instead we discover them. Perpetrator heuristic detection minimum number of change patterns, and patterns of trace, and the percentage of trace patterns in the patterns of change. Heuristic revealed on the largest possible number of patterns change, and patterns of trace, and the percentage of trace patterns.

Tracking mined patterns are not limited to bilateral patterns. Table 3 shows the minimum,

maximum, and average number of files in the trace detection training set patterns. Were detected trace patterns with up to seven files with CommitterDay heuristic. Now that our approach is able to find a possible trace patterns, there is a need to amount of "goodness" to evaluate the trace patterns. We validate our approach with three scales.

Table 3: The minimum, maximum, and average number of files in the traceability patterns in the training-set.

Heuristics Queries	Number of files		
	Min	Max	Average
Day	5,839	1,851	37.10
Committer	718	54	7.52
Committer Day	2,372	277	11.68

Let T= {ts_1, ts_2, .ts_m} and E = {es_1, es_2 ….es_n} be the training-set and assessment set correspondingly. Deliberate the pattern $es_i = \{f_1\} \rightarrow \{f_2\} \rightarrow \ldots \rightarrow \{f_k\}$ in the assessment set that is experiencing variations. To finally expect this pattern, the training-set can be enquired for candidates after each element {f_t} of the pattern es_i is different (or planned to be changed). Let $Ces_i = Ces_{i1} \cup Ces_{i2} \ldots . \cup Ces_{ik}$ where Ces_{ij} = {ts_1, tsp} be a set of candidate patterns recommended from the assessment set after altering the jth element (and previous elements) of the pattern es_i.

Definition: Covered pattern is a traceability-pattern in the assessment set for which present is at least one applicant pattern recommended from the training-set,

$$Covered\ Patterns = |\{\forall esi \in E \Rightarrow |Ces_i| > 0\}$$

Definition: Coverage is the proportion of the total number of covered patterns to the total number of patterns in the assessment set,

$$Coverage = \frac{CoveredPatterns}{|E|} \times 100\%$$

Definition: Correctly covered pattern is a covered pattern with at least one recommended applicant pattern from the preparation set that is the same (completely identical) or its sub-pattern (moderately identical).

Definition: Recall is the proportion of the total number of correctly covered patterns to the total number of patterns in the assessment set.

$$\text{Re}call = \frac{CorrectlyCoveredPatterns}{|E|} \times 100\%$$

Coverage and recall indicating the completion of the training set to predict the evaluation group. Describes how many of the trace patterns of coverage, the developer can expect that the recommended mined patterns in the training set. Describes mention how may of these recommendations are "correct." Ideally, it should be all of the coverage and call 100% (expected for patterns in the evaluation of a set correctly in the training set). Call coverage and give only one measure of the usefulness of tracking patterns to predict the change programs. Conduct arguably more important is how much of the total candidate patterns, both correct and incorrect, and suggested a range of training that require examination of the pattern covered in the assessment group.

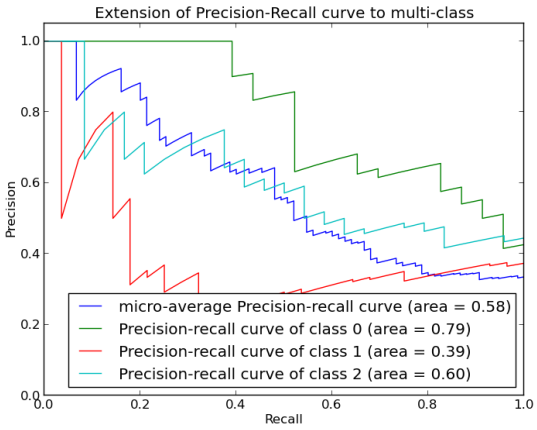

Fig. 4: Extension lead of Precision-Recall curve to multi-class.

Definition: The significance ratio of a covered pattern is the sum of the ratios of the number of pertinent patterns over the number of recommended candidates of all its basics. The significance ratio in our example is 2/3 + 1/2 + 1 = 2.167.

$$\text{Re}levance\ Ratio\ (esi) = \sum \frac{relevantPatterns}{|Ces_i|}$$

With regard to the internal validity, we are able to detect trace of change groups committed in KDE warehouse in different periods of time, patterns and sizes (not less than seven days) between 2005/05 and 2006/08. It also shows us an effective approach to the restoration of tracking links with only a few days (versions) from the date of the change (Vinárek, J., 2014). And our results are obtained from the date of the change of more than two dozen packets in KDE repository.

Our approach is also applied in the Apache httpd software repository and found similar results. Both systems are key examples of the successful development of open source, which represent areas of different sizes, but we do not claim that our results be generalized to any particular system (such as software development and commercially).

We believe that our approach is applicable to any software development practice with the commission of the relevant together in a software repository files. Also, check the validity of our results are only in the context of repeated patterns that are repeated, not every change, in the later part of history.

Conclusion and Future Works:

We cast-off versions the past to uncover traceability patterns involving of source code files and further artifacts. A heuristic centered method that uses common pattern mining is presented. The improved patterns give the detailed order in which the files in a pattern were deviations. We have shown that these links supports change tracking software to predict with high accuracy, whether a similar pattern occurred often in previous versions.

Our work compounded with the approach in tracking between applications in warehouses bug detection and source code expands the horizons Find tracking across mining software repositories in general and in general. While the debate here may seem limited to the open source development, and we believe that our approach applies equally in any other development methodology that exhibits different types of artifacts in the same change in groups. We see that the choice of the appropriate period of history, which increases the accuracy of the prediction software change is still a matter of investigation in the future. It also is being investigated in an additional inferred to collect change related groups such as the similarity of the text commit messages.

We sensation that mining at a better granularity for patterns (e.g., class or method level to paragraph) would produce improved results; at least in terms of recovering framework to developers. We are emerging tools in this way. Further, we proposal to take part our traceability tools directly into a version-control system.

REFERENCES

Ali, N., 2012. *Analysing Source Code Structure and Mining Software Repositories to Create Requirements Traceability Links* (Doctoral dissertation, École Polytechnique de Montréal).

Ali, N., A. Sabané, Y.G. Guéhéneuc, G. Antoniol, 2012. Improving Bug Location Using Binary Class Relationships. In *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on* pp: 174-183. IEEE.

Ali, N., F. Jaafar, A.E. Hassan, 2013. Leveraging historical co-change information for requirements traceability. In *Reverse Engineering (WCRE), 2013 20th Working Conference on* pp: 361-370. IEEE.

Ali, N., Y. Guéhéneuc, G. Antoniol, 2011. Trust-based requirements traceability. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* pp: 111-120. IEEE.

Ali, N., Z. Sharafi, Y.G. Guéhéneuc, G. Antoniol, 2014. An empirical study on the importance of source code entities for requirements traceability.*Empirical Software Engineering*, 1-37.

Bavota, G., A. De Lucia, R. Oliveto, G. Tortora, 2014. Enhancing software artefact traceability recovery processes with link count information. *Information and Software Technology*, *56*(2): 163-182.

Borg, M., P. Runeson, A. Ardö, 2013. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability.*Empirical Software Engineering*, 1-52.

Capobianco, G., A.D. Lucia, R. Oliveto, A. Panichella, S. Panichella, 2013. Improving IR-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, *25*(7): 743-762.

Corley, C.S., N.A. Kraft, L.H. Etzkorn, S.K. Lukins, 2011. Recovering traceability links between source code and fixed bugs via patch analysis. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering* pp: 31-37. ACM.

Cuddeback, D., A. Dekhtyar, J.H. Hayes, 2010. Automated requirements traceability: The study of human analysts. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pp: 231-240. IEEE.

Cuddeback, D., A. Dekhtyar, J.H. Hayes, J. Holden, W.K. Kong, 2011. Towards overcoming human analyst fallibility in the requirements tracing process (NIER track). In *Proceedings of the 33rd International Conference on Software Engineering*, pp: 860-863. ACM.

De Lucia, A., A. Marcus, R. Oliveto, D. Poshyvanyk, 2012. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability*, pp: 71-98. Springer London.

De Lucia, A., M. Di Penta, R. Oliveto, A. Panichella, S. Panichella, 2013. Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation. *Information and Software Technology*, *55*(4): 741-754.

De Lucia, A., M. Di Penta, R. Oliveto, A. Panichella, S. Panichella, 2011. Improving ir-based traceability recovery using smoothing filters.

In*Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* pp: 21-30. IEEE.

Dekhtyar, A., O. Dekhtyar, J. Holden, J.H. Hayes, D. Cuddeback, W.K. Kong, 2011. On human analyst performance in assisted requirements tracing: Statistical analysis. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pp: 111-120. IEEE.

Gethers, M., B. Dit, H. Kagdi, D. Poshyvanyk, 2012. Integrated impact analysis for managing software changes. In *Software Engineering (ICSE), 2012 34th International Conference on* pp: 430-440. IEEE.

Gethers, M., R. Oliveto, D. Poshyvanyk, A.D. Lucia, 2011. On integrating orthogonal information retrieval methods to improve traceability recovery. In Software Maintenance (ICSM), 2011 27th IEEE International Conference on pp: 133-142. IEEE.

Ghafari, M., C. Ghezzi, A. Mocci, G. Tamburrelli, 2014. Mining unit tests for code recommendation. In *ICPC*, pp: 142-145.

Jaafar, F., Y.G. Guéhéneuc, S. Hamel, G. Antoniol, 2014. Detecting asynchrony and dephase change patterns by mining software repositories.*Journal of Software: Evolution and Process*, *26*(1): 77-106.

Kong, W.K., J. Huffman Hayes, A. Dekhtyar, J. Holden, 2011. How do we trace requirements: an initial study of analyst behavior in trace validation tasks. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pp: 32-39. ACM.

Panichella, A., C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, A. De Lucia, 2013. When and how using structural information to improve ir-based traceability recovery. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on* pp: 199-208. IEEE.

Qusef, A., G. Bavota, R. Oliveto, A. De Lucia, D. Binkley, 2014. Recovering test-to-code traceability using slicing and textual analysis. *Journal of Systems and Software*, *88:* 147-168.

Qusef, A., R. Oliveto, A. De Lucia, 2010. Recovering traceability links between unit tests and classes under test: An improved method. In *Software Maintenance (ICSM), 2010 IEEE International Conference on* pp: 1-10. IEEE.

Thomas, S.W., A.E. Hassan, D. Blostein, 2014. Mining Unstructured Software Repositories. In *Evolving Software Systems* pp: 139-162. Springer Berlin Heidelberg.

Vinárek, J., P. Hnětynka, V. Šimko, P. Kroha, 2014. Recovering Traceability Links Between Code and Specification Through Domain Model Extraction. In *Enterprise and Organizational Modeling and Simulation*, pp: 187-201.