ImpactMiner: A Tool for Change Impact Analysis

Bogdan Dit, Michael Wagner, Shasha Wen, Weilin Wang, Mario Linares-Vásquez, Denys Poshyvanyk, and Huzefa Kagdi*

Computer Science Department
The College of William and Mary
Williamsburg, VA 23185, United States
{bdit, mmwagn, swen, wwang01, mlinarev,denys}@cs.wm.edu

*Department of Computer Science
Wichita State University
Wichita, KS 67260-0083, United States
kagdi@cs.wichita.edu

ABSTRACT

Developers are often faced with a natural language change request (such as a bug report) and tasked with identifying all code elements that must be modified in order to fulfill the request (*e.g.*, fix a bug or implement a new feature). In order to accomplish this task, developers frequently and routinely perform change impact analysis. This formal demonstration paper presents ImpactMiner, a tool that implements an integrated approach to software change impact analysis. The proposed approach estimates an impact set using an adaptive combination of static textual analysis, dynamic execution tracing, and mining software repositories techniques. *ImpactMiner* is available from our online appendix http://www.cs.wm.edu/semeru/ImpactMiner/

Categories and Subject Descriptors

D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement – enhancement, restructuring, reverse engineering, and reengineering

General Terms

Documentation, Design

Keywords

Change impact analysis, repository mining, subversion, dynamic analysis, information retrieval

1. INTRODUCTION

Throughout the development and maintenance of a software system, developers are frequently given change requests that they must implement in the project. Given such a change request (*e.g.*, bugs or feature requests) the developers must identify relevant source code entities that need to be modified. This activity is referred to as change impact analysis or simply **i**mpact **a**nalysis (IA) [1] and it is a fundamental part of the software development and maintenance process.

Two good starting points in performing IA are project's documentation and developers' experience on that project. However, many times these sources are not available, since the documentation for a given project may be either inaccurate or missing entirely or because the original developers are no longer available. In such situations, developers must look into other sources of information about the project. IA was traditionally performed by analyzing the static and dynamic information about a software system, or by using Information Retrieval (IR) techniques. In addition to these traditional sources of information, one recent source of information that is recognized and used is the software repositories. Mining Software Repositories (MSR) approaches analyze historical changes in the source code repositories to determine co-changed source code elements and infer association rules. Gethers *et al.* [2] proposed a scenario driven approach for IA that recommends impact sets based on the available sources of information generated using IR, dynamic and MSR techniques.

In this paper we introduce a novel tool for IA called *ImpactMiner*, which implements the adaptive approach to IA that was evaluated in our previous work [2]. ImpactMiner is a new addition to the limited suite of IA tools, such as JRipples [3] and Chianti [4], which use static and/or dynamic techniques. ImpactMiner uses co-changes from source code repositories, as well as, combining multiple sources of information. Our previous work [2] showed that the combination of the results are statistically significantly better than the results obtained from any single technique or any combination of two techniques.

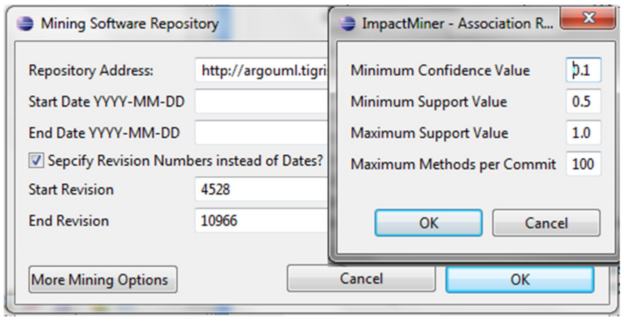
2. BACKGROUND AND RELATED WORK

2.1 Integrated Impact Analysis

Our approach adapts to the available sources of information, such as textual, dynamic and historical, and provides support for combining these sources of information to improve the results for IA [2]. For a given maintenance task the approach works as follows. First, IR is used to retrieve textually similar methods for a change request. Second, if the developer is able to generate an execution trace based on a scenario that exercises the feature of interest, she can combine this information with the textual information (similar to the SITIR approach [5]) in order to get improved results. Finally, once the developer identifies a relevant method, she can use historical information to get recommendations on other relevant methods (*i.e.*, methods that could be of interest because they were co-changed frequently in the past). In addition, she can use the historical information in combination with textual and dynamic information (if available). An evaluation involving four Java systems showed that combining IR, dynamic and MSR techniques produced results that were statistically significant better than stand-alone techniques [2].

2.2 Related Work

Sando [6] is an open-source source code search tool implemented as a Visual Studio extension. In addition, Sando [6]



Repository Address: http://argouml.tigri
Start Date YYYY-MM-DD
End Date YYYY-MM-DD
☑ Sepcify Revision Numbers instead of Dates?
Start Revision 4528
End Revision 10966

Minimum Confidence Value 0.1
Minimum Support Value 0.5
Maximum Support Value 1.0
Maximum Methods per Commit 100

Figure 1 Options for Mining Software Repositories (left) and for Associations Rule Mining (right)

provides an extensible framework that can be easily adapted and configured to support new search techniques. JRipples [3] is an Eclipse plug-in that uses static information to analyze dependencies between entities in order to help developers locate the impact set manually, by keeping track of visited elements and the elements that dependent on them. Chianti [4] is also an Eclipse plug-in that uses static information, in the form of changes between consecutive versions of the software, and dynamic information, such as execution traces of test cases. By combining these two types of information, Chianti is able to predict a set of unit tests that were affected by the changes in the source between two versions.

ImpactMiner is built on top of FLAT[3] [7], a tool for performing feature location using textual searches, execution traces, a combination of the two, annotating features, and visualization. ImpactMiner is different than JRipples and Chianti because in addition to static and dynamic information, it leverages historical data and enables developers to combine these types of information in the way that bests suits their needs.

We refer the interested reader to the related work presented in our previous work [2].

3. IMPACTMINER

ImpactMiner is implemented as an Eclipse plug-in on top of FLAT[3] [7]. In addition to the inherited features from FLAT[3], ImpactMiner extracts evolutionary co-changes from SVN repositories, and allows developers to perform IA via the techniques introduced by Gethers *et al.* [2], namely IR, dynamic (Dyn) and historical (Hist) techniques, or any combinations of these. Among these three techniques IR is the least costly in terms of human effort, but its performance is not always optimal [2]. Using additional Dyn or Hist sources of information requires more human effort in terms of gathering the data, but it produces more accurate results. In order to address the tradeoff between human effort and performance we implemented ImpactMiner to allow developers to share the data produced by the Dyn and Hist techniques with other developers. For example, a manager or a project leader can generate a set of execution traces or can mine data from the SVN repositories and extract evolutionary couplings, and can share this data with the rest of the team. The benefit consists of reducing the amount of manual human effort to a minimum.

3.1 Inherited Features

In addition to its own original features, ImpactMiner inherits the following features from FLAT[3] that were used for feature location: using the IR engine based on Lucene[1] to search for methods given a natural language user query, generating execution traces, combining results from IR and execution traces, annotating features, and visualizing the distribution of the search results among different lines of code.

One of the valuable inherited features to Impact Analysis is feature annotation. Having identified one or more methods as being related to a high-level concept or feature, the developer can annotate those methods with the name of the feature (or bug ID) that they are related to. This annotation should simplify the future attempts to fix or improve such a feature, as there will already be an identified set of related methods.

3.2 Extracting Evolutionary Co-Changes

ImpactMiner supports IA by extracting evolutionary co-changes from SVN repositories. The major preprocessing steps are enumerated next.

3.2.1 SVN Repositories Mining

The first part of generating evolutionary co-changes consists of mining SVN repositories to extract change-sets of the source code. Figure 1 (left) shows the options needed to mine the repository, such as the URL of the SVN repository (*e.g.*, the trunk or any repository branches if needed). We use the SVNKit[2] library for all the operations involving the SVN repository. By default, the access to the repository uses anonymous authentication. However, in cases when repositories require authentication, a window prompting for a username and password will appear. In addition, developers must select a period of time for extracting SVN commits, by specifying a start and end date. Alternatively, the developer can choose a more precise range of revisions, by specifying the starting and ending revisions (see Figure 1 (left)).

ImpactMiner iterates through the SVN commits specified by the developer, and stores the java source code files associated with those commits on disk. If a file was added in revision N, the file stored on disk is in folder $N/fileName.java.revN$. If a file was modified in revision N, then that version of the file and the previous version of the file are saved on disk (*e.g.*, $N/fileName.java.revN$ and $N/fileName.java.revN-1$).

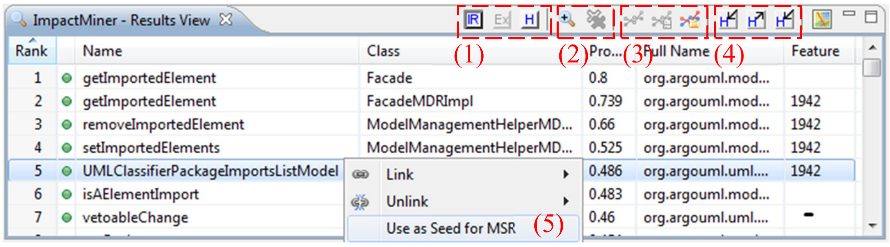
The developer has the option to refine the range of revisions (see Figure 2 (4)), in order to include more or less history to extract evolutionary couplings. For systems with abundant history, downloading a large period of history can take from a few minutes to a few hours. To alleviate this potential issue, ImpactMiner supports caching and importing/exporting of change-sets. The caching mechanism allows all the previously downloaded change-sets to be stored on disk. Hence, if the developer decides to increase the range of commits, only the change-sets that were not previously downloaded will need to be downloaded, reducing the waiting time considerably. The import/export features (see Figure 2 (4)) allow sharing the downloaded data with other team members.

3.2.2 Generating Itemsets

The second part of generating evolutionary co-changes consists of extracting the names of the methods that were modified in the SVN commits. We used the java tool made available by Dit *et al.* [8], which uses the Eclipse Java Development Tools (JDT)[3] to generate the generate an abstract syntax tree (AST) of a java class.

For each of the downloaded SVN commits, if a file was modified in that commit, (i) we generated an AST of the current file version, (ii) we generated an AST of the previous file version and (iii) we compared the current and previous version in order to

[1] http://lucene.apache.org/
[2] http://svnkit.com/
[3] http://www.eclipse.org/jdt/



Rank		Name	Class	Pro...	Full Name	Feature
1	●	getImportedElement	Facade	0.8	org.argouml.mod...	
2	●	getImportedElement	FacadeMDRImpl	0.739	org.argouml.mod...	1942
3	●	removeImportedElement	ModelManagementHelperMD...	0.66	org.argouml.mod...	1942
4	●	setImportedElements	ModelManagementHelperMD...	0.525	org.argouml.mod...	1942
5	●	UMLClassifierPackageImportsListModel		0.486	org.argouml.uml....	1942
6	●	isAElementImport		0.483	org.argouml.mod...	
7	●	vetoableChange		0.46	org.argouml.uml....	-

Figure 2 The Results View of ImpactMiner: The columns represent the rank, accessibility levels, name of the method/field, class name, IR similarity or confidence value, full method name, feature name. (1) The buttons represent the three states (*e.g.*, disabled, hidden and enabled) for the three sources of information: textual (IR), dynamic (Ex) and historical (H). (2) IR buttons to refine and clear the query. (3) Buttons for running/exporting/ importing a trace. (4) Buttons for configuring MSR settings, exporting and importing the MSR data; (5) Option to run MSR technique using the selected method as seed

produce a list of methods that were modified between those two versions. If a file was added to an SVN commit, we used the same tool to extract its method names. Note that commits with a large number of modified files were excluded from the analysis.

The list of modified methods, which are associated with each revision, constitute the *itemsets* we will use as input for finding the frequent itemsets in the *transactions* (*i.e.*, SVN commits). These itemsets are cached on disk to avoid regenerating them once they are generated. In addition, they can be shared between team members using the import/export feature (see Figure 2 (4)).

3.2.3 *Mine Evolutionary Co-Changes*

Once the transactions (*i.e.*, SVN commits) and their itemsets (*i.e.*, list of methods changed in those commits) are generated, we mined *association rules* of the form $\{mA\} \Rightarrow \{mB, mC, \ldots\}$. The association rules that have less than a user specified support and confidence value are ignored. The support value of an itemset X, noted as $supp(X)$, is characterized by the proportion of transactions which contain that itemset. The confidence of a rule $X \Rightarrow Y$ is defined as $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$. The developer can specify the minimum values for support and confidence in the settings window (see Figure 1 (right)).

These rules have only one method as the antecedent (*e.g.*, mA), which represents the seed method (*e.g.*, see method *UMLClassifierPackageImportsListModel* from Figure 2 and Figure 2 (5)), and a set of methods as consequent (*e.g.*, mB, mC, etc.). Intuitively, these rules can be interpreted as "developers who modified method mA in the past, also modified methods mB, mC, etc.". ImpactMiner takes as input a seed method that a developer found to be of interest, and suggests a list of methods that could be of interest for IA. These methods represent the union of all the consequents, where the seed method is the antecedent, ranked by their confidence and support values. These results are presented to the user in the "Results View" (see Figure 3, where the results of Hist are enabled, results or IR are hidden, and results of Dyn are disabled. These methods are ranked based on their confidence scores).

Similarly to the other historical data, the association rules that are generated for a given period and a given support and confidence values are cached on disk and can be shared with other developers.

3.3 IR, Dyn and Hist

Based on the types of data available, ImpactMiner allows the developer to modify or refine the data produced by those techniques (*e.g.*, IR, Dyn or Hist), analyze the results of individual techniques, or combine the results of those techniques to achieve better results [2].

3.3.1 *Use or Modify Data from Individual Techniques*

At any given point, the developer can choose to use one of the three techniques in order to get a list of recommended methods for IA.

She can define a query to obtain a list of textually similar methods and fields to the query. If she is not satisfied by the results produced by Lucene, she can refine the query to obtain other results (see Figure 2 (2)).

The developer can use dynamic information by collecting an execution trace of the software system. By choosing the trace button (see Figure 2 (3)) she can collect traces until she obtains satisfactory results. In addition she can export the collected trace and share it with someone else, or she can import a trace that someone else produced.

The developer also has the option to generate association rules by specifying the repository address, the time period and the parameters for the association rule algorithm (*e.g.*, support and confidence) (see Figure 1 (right)). She can generate new association rules by modifying the amount of history and the parameters at any time (see Figure 1). In addition, she can choose different seeds (*i.e.*, starting methods) as antecedent, to get different results produced by the MSR technique. For example, in the "Results View" once she identified a method with potential interest, she can choose the option "Use as seed for MSR" (see Figure 2) to get a list of methods that were modified in the same commits in the past. This option is similar to the "Open Call Hierarchy" option provided by Eclipse IDE, which for a selected method, provides the callers and called methods.

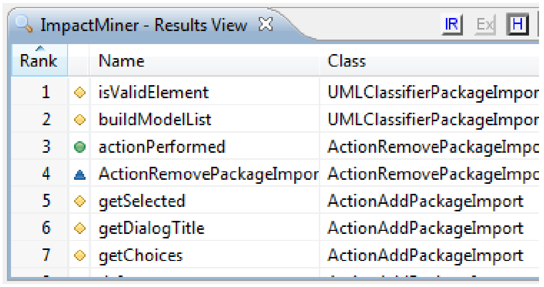
3.3.2 *Analyze Data from Individual Techniques*

The "Results View" (see Figure 2 (1)) contains three buttons that correspond to the results from IR (IR), Dyn (Ex) and Hist (H). These buttons have three states: *disabled*, if there is no information associated with them, *hidden*, if the information is not shown in the view or *enabled*, if the information is presented in the view. In Figure 2, IR is enabled, Dyn is disabled (*i.e.*, there is no execution trace), and Hist is hidden.

Note that the results of a technique are displayed in the view only when one of those buttons is set to *enabled*, and the other two are set to either *disabled* or *hidden*. If more than one button is set to *enabled*, the results displayed will be the ones from the combination of the results generated by their associated techniques.

3.3.3 *Combining IR, Dyn and Hist*

By enabling the buttons associated with the results produced by the individual IR, Dyn and Hist techniques (Figure 2 (1)), the developer can analyze the results of their combination. Note that the results of IR are always ranked based on their similarities to the user



ImpactMiner - Results View

Rank		Name	Class
1	◆	isValidElement	UMLClassifierPackageImpor
2	◆	buildModelList	UMLClassifierPackageImpor
3	●	actionPerformed	ActionRemovePackageImpc
4	▲	ActionRemovePackageImpor	ActionRemovePackageImpc
5	◆	getSelected	ActionAddPackageImport
6	◆	getDialogTitle	ActionAddPackageImport
7	◆	getChoices	ActionAddPackageImport

Figure 3 Results of generated by the MSR technique when the method UMLClassifierPackageImportsListModel was used as a seed

query and the Hist results are ranked based on their confidence and support values. The results produced by the Dyn technique are not ranked. For additional details about the combination process we refer the interested reader to our previous work [2].

IRDyn Enabling both the IR (IR) and Dyn (Ex) buttons the results produced are the ones generated by the SITIR [5] approach (*i.e.*, it eliminates from the IR results the ones that do not appear in the execution trace).

IRHist By combining these two approaches, the results produced by IR on positions i appear in the IRHist list on positions $2i - 1$, whereas the results produced by Hist on positions i appear in the IRHist list on positions $2i$. If a method appears in both IR and Hist lists, it will appear only once in IRHist.

DynHist This combination returns only the methods that appear both in the execution trace and in the results of the association rules.

IRDynHist The combination of all three sources of information is performed as follows. First, the results of IRDyn are computed. Second, the results of IRDyn are interweaved with the results of Hist (*i.e.*, the by IRDyn on positions i appear in the IRDynHist list on positions $2i - 1$, whereas the results produced by Hist on positions i appear in the IRDynHist list on positions $2i$). Similarly to the case of IRHist, if a method appears in both IRDyn and Hist lists, it will appear only once in IRDynHist.

3.4 Availability

More information about ImpactMiner can be found on our webpage[4], which contains the source code and a video demonstrating its main features.

3.5 Usage Example

Our approach to IA instantiated in our ImpactMiner tool was evaluated in our previous work [2]. In this section we provide an example of the results by ImpactMiner for issue #1942 of ArgoUML[5]. The results produced using IR are presented in Figure 2, and the results produced by choosing the method on rank five as the seed method for Hist are presented in Figure 3. The results returned by Hist are part of the gold set associated with bug #1942 and are complementary with the results returned by IR (see column "Feature" from Figure 2).

3.6 Future Work

First we want to include the incremental indexing functionality of Lucene, which will allow updating the corpus with only the methods that were changed, as opposed to re-indexing the entire system for each change in the code. Second, we want to provide statistics such as the number of commits downloaded, average number of commits per file, number of transactions, average number of itemsets, etc. This will help the user choose the association rules parameters more easily. Third, we want to improve our export/import feature for all the historical data (*e.g.*, commits, itemsets, association rules), to make sharing easier. Fourth, we want to provide users with different options for combining the results from these three different sources. Fifth, we want extend ImpactMiner to support GIT repository mining in addition to SVN repository mining. Finally, we would like to conduct a user study to evaluate the benefits of this tool.

4. CONCLUSIONS

We introduced ImpactMiner, a tool that implements the adaptive approach to IA presented in our previous work [2]. ImpactMiner allows developers to extract co-changes from SVN repositories at method level granularity. In addition it allows developers to use different techniques (*e.g.*, IR, dynamic and MSR), or a combination of these techniques, based on the available sources of information.

5. ACKNOWLEDGMENTS

We would like to thank Malcom Gethers for his contribution to the IA approach that ImpactMiner is based on. This work is supported by NSF CCF-1016868, NSF CCF-1156401 and NSF CCF-1253837 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

6. REFERENCES

[1] S. Bohner and R. Arnold, *Software Change Impact Analysis*. Los Alamitos, CA: IEEE Computer Society, 1996.

[2] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated Impact Analysis for Managing Software Changes," in *34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, 2012, pp. 430-440.

[3] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: A Tool for Program Comprehension during Incremental Change," in *13th IEEE International Workshop on Program Comprehension (IWPC'05)*, St. Louis, Missouri, USA, 2005, pp. 149-152.

[4] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a Tool for Change Impact Analysis of Java Programs," in *19th Conference on Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA '04)*, Vancouver, BC, Canada, 2004, pp. 432-448.

[5] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace," in *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, Atlanta, Georgia, 2007, pp. 234-243.

[6] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: An Extensible Local Code Search Framework," in *20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE'12)*, 2012, pp. 1-4.

[7] T. Savage, M. Revelle, and D. Poshyvanyk, "FLAT^3: Feature Location and Textual Tracing Tool," in *32nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*, Cape Town, South Africa, 2010, pp. 255-258.

[8] B. Dit, A. Holtzhauer, D. Poshyvanyk, and H. Kagdi, "A Dataset from Change History to Support Evaluation of Software Maintenance Tasks," in *10th Working Conference on Mining Software Repositories (MSR'13), Data Track*, San Francisco, CA, 2013, pp. 131-134.

[4] http://www.cs.wm.edu/semeru/ImpactMiner/

[5] http://argouml.tigris.org/issues/show_bug.cgi?id=1942