Quantifying and Assessing the Merge of Cloned Web-Based System: An Exploratory Study

Jadson Santos
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte, UFRN
Natal, Brazil
jadsonjs@gmail.com

Uirá Kulesza
Department of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte, UFRN
Natal, Brazil
uira@dimap.ufrn.br

Abstract— **This paper presents an exploratory study that analyzes the complexity to integrate existing merge conflicts of a cloned large-scale web system. The study is supported by an existing tool that focuses on the identification of merge conflicts that can arise during the integration of cloned systems. The approach addresses the merge conflict analysis through the extraction and comparison of the issues and code history of cloned systems using mining software repository and static analysis techniques. The main aims of our study are: (i) to quantify the kind of conflicts defined by our approach that happen when evolving cloned systems; (ii) to evaluate if they are being correctly detected by our tool; and finally (iii) to analyze the difficult to integrate them from one cloned system to another. The study findings show: (i) a predominance of semantic conflicts between issues of source and target cloned systems; and (ii) the feasibility to use merge analysis approaches to integrate tasks from one clone to another.**

Clone-and-own approach, web-based information systems, software merge, code merge conflicts

I. Introduction

Modern software development involves the parallel working of software developers, who work separately on creating and modifying their local copies of code assets, and then need to submit them to version control repositories. In this context, software merge techniques are used to promote the integration of code assets modified in parallel by different developers. When merging code to repositories, conflicts naturally emerge due to code change operations on the same code elements – such as classes or methods, and also because of semantic dependencies between the modified classes [1] [2] [3] [4].

Recent research work [3] [4] [5] has investigated and proposed approaches for the detection and analysis of code merge conflicts. Zimmermann et al [5] analyzed four open source systems from CVS repositories and found a substantial amount of textual merging conflicts – 23% to 43% – in those projects. Brun et al [3] investigated nine active open source projects and found that 16% of the analyzed merge operations contain textual conflicts and a significant number presents high-order conflicts – related to changes semantically incompatible that causes compilation or test errors. Guimarães & Silva [4] propose an automated approach that continuously analyzes and detects merge conflicts from committed and uncommitted changes with the main aim to anticipate and present them to software developers. Their approach works with a rich set of different merge conflicts.

In addition, Dubinsky et al [6] conducted an exploratory study to investigate the cloning culture in six industrial product lines. They observed that companies usually clone existing products with the main aim of addressing new requirements and customize those products to new contexts and scenarios. In their study, they conclude that cloning is considered a favorable reuse approach that facilitates independent customization of new products based on existing ones. However, they also noticed that the cloning practice brings difficulties when performing maintenance and evolution activities, such as propagating changes between those clones, and integration of the cloned code assets. The authors suggest that clone management techniques could be explored in future research work.

While recent empirical studies have already given a perspective of the different merge conflicts that happen in existing open source systems, there is no empirical study providing a quantitative and qualitative detailed view of how those conflicts are happening and the complexity to integrate them in the context of cloned commercial systems. In addition, the understanding of the complexity of merging cloned systems and the development of techniques to help this activity is also of interest of the software product line community, which has recently identified [6] that cloned techniques are used to manage variabilities from a software product line (or software family).

In this context, this paper presents an exploratory study that quantifies and analyzes the complexity to integrate existing merge conflicts of a cloned large-scale web system. The main aim of our study is: (i) to quantify the three kind of conflicts – structural, semantic and lexical – defined by our approach that happen when evolving cloned systems; (ii) to evaluate if those conflicts are begin correctly detected by our tool; and (iii) to analyze the difficult to integrate them from one cloned system to another. The study is supported by an existing tool [10] that focuses on the identification of merge conflicts that can arise during the integration of cloned systems. The approach

DOI reference number: 10.18293/SEKE2016-232

addresses the merge conflict analysis through the extraction and comparison of the issues and code history of cloned systems [7] using mining software repository and static analysis techniques. The contributions of our study are as follows: (i) it performs a systematic characterization and analysis of the kind and complexity of merge conflicts for large-scale web-based systems; and (ii) it shows the feasibility to use merge analysis approaches to integrate tasks from one clone to another.

The rest of this paper is organized as follows. Section II describes the study settings. Section III presents the study results. Section IV discusses threats and limitation of our study results. Section V reports the related work. Finally, Section VI presents concluding remarks and future work.

II. Study Settings

The main aims of our study are: (i) to understand the kind of conflicts that happen when evolving cloned web-based systems; (ii) to evaluate the conflict detection by our tool; and (iii) to analyze the complexity to integrate issues developed for one cloned system – called the source system – to another one – the target system.

A. *Categorization of Code Merge Conflicts*

In the first step, we considered a categorization of code merge conflicts based on existing research work [2] [3] [4]. A merge conflict is a pair of code changes developed for the source and target cloned systems, which interfere each other when merging code changes from the source to the target. In our study we have focused on the following kinds of conflicts that happen in the context of 3-way merging [1] – which uses information from a common ancestor besides the one available for the two classes being merged:

i. *direct conflict* (structural) – represents a pair of code changes applied to the same code elements (e.g., attributes, methods) by both source and target systems;

ii. *indirect conflict* (semantic) – happens when code changes applied to the source system are in the call graph of other code changes in the target system; and

iii. *pseudo conflict* (lexical) – the source and target systems modify the same class or interface, but different and independent code elements (attribute or method).

We quantify pseudo conflicts only to understand the additional effort of developers when merging cloned systems using textual merge tools. Textual-based tools usually exhibit these conflicts, but the usage of more advanced merge tools (e.g., analysis of direct conflicts) can avoid this additional effort.

B. *Data Collection and Analysis Procedure*

To support the analysis of conflicts in the merge process, we performed the following data collection procedures, which were automated by a tool we developed:

Step 1: Mine Evolution History. First, the MergeClear tool mines the development issues (change requests) of the cloned web systems from issue tracking systems (such as Bugzilla) used by each institution. It produces as output the task evolution history file, an XML file (one for the source and another one for the target system) that contains all the development issues (with information such as description, version, kind of issue, related modules, etc), and respective code revisions associated to them.

Step 2: Mine Code Evolution. After that, the tool recovers and compares subsequent revisions associated to the Java code assets mined from version control systems to extract fine-grained code change operations applied to them. In particular, in this study, we focused on the following code change operations: addition, deletion and modification of classes, attributes and methods. Our tool can currently also quantify code change related to class inheritance, interface implementation, and code annotations, but they were not explored in our study. All this information is stored in the change log history file, which is a refinement of the issue evolution history.

Step 3: Analysis of Merge Conflicts. Next, the tool processes and compares these change log files produced for the source and target cloned web-based systems to automatically calculate the direct, indirect, and pseudo conflicts between them. The direct conflicts are quantified by identifying attributes, methods and classes that were modified in both source and target systems. The computation of indirect conflicts also utilizes the system call graph to quantify which of the code change operations in the source system are in the call graph of other code change operations from the target system. Pseudo conflicts are calculated by identifying code change operations applied to the same classes and do not have indirect conflicts. The tool can also be used to visualize the list of development issues and associated code changes applied to the source and target web-based systems. In addition, we also present the list of code conflicts that will occur when merging the development issues of the cloned web-based systems with their respective conflicting code changes.

Step 4: Group and Order Issues. After the conflict analysis step, the issues were ordered and grouped to represent different integration scenarios. The main criteria were (i) the kind of the issues (ii) the kind of conflicts detected in the issue; (iii) the amount of source code artifacts changed in the issues; and (iv) if the issues change different layers or modules of the system or if they only modify restricted code.

Step 5: Issues Selection. After the issue classification, a specific set of issues was selected to represent different integration efforts. In our study, we have not analyzed issues with pseudo conflicts because they do not represent real conflicts when merging code from source to the target system [12]. For all the selected issues, we used the merge functionality of the subversion plugin to integrate code changes from the source to the target system and compared with the results indicated by our tool. Fig. 1 illustrates this analysis workflow.

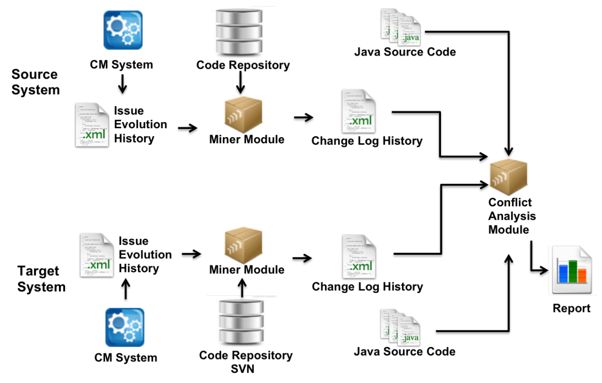


Figure 1: Automated Approach to Reconciliation of Cloned Systems

After the tool execution, we conducted a study guided by the following research questions:

- RQ1. Which amount and kind of code merge conflicts happen when evolving and merging cloned web-based systems?
- RQ2. Do issues without merge conflicts can really be automatically integrated?
- RQ3. Are direct conflicts correctly identified? How complex they are to be integrated?
- RQ4. Are indirect conflicts correctly identified? Do the indirect conflicts can cause behavior problems on the target system as the approach suggest?

C. *Target Web-based System*

The Federal University of Rio Grande do Norte (UFRN, Brazil) develops a set of web information systems designed to automate business processes for universities focusing on different and complementary aspects, such as academic, administration, planning and management. These systems began to be deployed in 2006. Our study focuses on one of these systems. Table I shows an overview of the system in terms of users and size.

TABLE I. SIZE AND USERS OF ANALYZED SYSTEM

System	Total of Users	Daily Access	KLOC	Java Classes	Java Methods
SIGAA	52000	56000	833	5906	81102

The SIGAA system was chosen because it is a large-scale web-based information system implemented in the Java language. The system uses the clone-and-own strategy to promote its customization to other partners. Also, we have complete access to the version control and issue tracker systems of the development teams, responsible for the source and target systems. The scenario chosen for this study involved a total of 1083 issues.

III. STUDY RESULTS

This section presents and discusses the obtained results for our research questions.

A. *RQ1: Which amount and kind of code conflicts happen when evolving and merging cloned web-based information systems?*

We analyzed the amount and kinds of conflicts considering the perspectives of development issues and code change operations. Table II displays the distribution of conflicts detected by our tool in the analyzed issues. 65,18% of issues presents no conflict. 5,6% of issues had only direct conflicts. 18,83% of issues held no direct conflicts but present at least one indirect conflict, and 10,34% of issues have only pseudo conflicts.

Table III shows the amount of code conflicts that were calculated for the merge of the investigated cloned web systems. Most of the collected code conflicts (68,14%) are indirect. It means that a great number of semantic conflicts that happen when merging web information systems are not made explicit when only using textual-based merging tools. Developers should be aware of them to understand the change impact of the merging, and to identify specific parts of the system – the affected call graphs – to be re-tested after the integration. Our study also found a significant number of pseudo conflicts (22.20%). They represent additional useless effort from the developers when merging classes with existing textual-based tools, because those tools are not able to identify independent code changes applied to the same classes. Finally, Table III shows that a reduced number of direct conflicts (9,65%) were found in our study compared to the other kinds. Those direct conflicts require the developer intervention to be merged from the source to the target web-based system, because they represent direct changes applied to the same code elements (attributes, methods).

TABLE II. AMOUNT OF ISSUES BY KIND OF CONFLICT

	Without Conflicts	With Direct Conflict	With Indirect Conflict	With Pseudo Conflict
Amount of Issues	706 (65,18%)	61 (5,6%)	204 (18,83%)	112 (10,34%)

TABLE III. KINDS OF CONFLICTS DETECTED

Kind of Conflicts	Direct Conflict	Indirect Conflict	Pseudo Conflict
SIGAA	120 (9.65%)	847 (68.14%)	276 (22.20%)

B. *RQ2: Do issues without merge conflicts can really be automatically integrated?*

This research question focuses on evaluating if issues that were classified as having no conflicts by our approach can be automatically integrated using the traditional merge mechanisms from existing control version systems.

In order to answer this research question, we have selected a total of 15 issues between those ones identified as having no merge conflicts by our approach (Table II). After that, we identified the different commits and respective code changes associated with each issue, and they were manually applied from source to the target cloned web system using the merge functionality of the subversion plugin. Finally, we verified for each issue if there was no compilation error after the code merge of the different commits associated to the issue.

Table IV shows the results of our analysis. We have found that 11 issues of the source cloned web system from the 15 analyzed (73%) could be integrated without causing any compilation errors in the target web system.

TABLE IV. ANALYSIS RESULT OF THE RQ2

Analyzed Issues	Possible to Integrate	Dependence Error	Mining Error	No Manual Merge Possible
15	11 (73.33%)	1 (6.6%)	1 (6.6%)	2 (13.33%)

During our analysis we have observed that one issue exhibited a dependence error, which indicated that this issue could only be integrated if another one is integrated previously. It means that only the criterion of conflicts between code artifacts was not enough to ensure that the integration occurred without adding compilation errors in the target system. Because of that, it was also necessary to analyze the dependencies between issues. Thus, an issue from the source system could be integrated without compilation errors in the target system only if it has no conflicts and all the issues that it depends on were integrated before it.

The dependency computation algorithm used in our study is based on the data of the issue creation. This criterion was not enough to determine whether an issue has been implemented before another one or not. In some cases it is not trivial to determine whether an issue was accomplished before another one, because they occur in parallel, having interleaved commits. This has generated a mining error during the integration of one issue (Table IV) and it needs to be improved in the conflict analysis module.

Finally, there are two issues from the source system, which are not possible to merge in the target web system (Table IV). In one issue, some classes have been changed, but part of this evolution was registered in another issue. Thus, it brought difficulties to manually separate the source code belonged to the specific issue before proceed with the automatic merge. The second issue received the first commit in 2010, but the issue was not totally finished. Only in 2012, the issue was completed. Meanwhile, several commits were registered which originated many code changes for other issues, thus making very complex to perform a manual analysis of the issue, and separating just its specific changes. The discovering of such cases revealed the need to correctly register the association between issues and respective commits in the issue tracker or control version systems.

C. RQ3: Are direct conflicts correctly identified? How complex they are to be integrated?

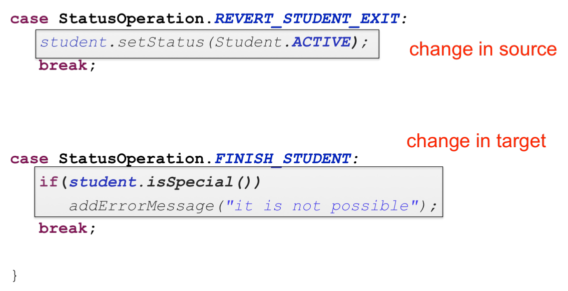
In this research question, we have analyzed the direct conflicts identified by MergeClear tool in order to determine whether they were correctly identified and how complex they are to be integrated from the source to the target system. In short, we have analyzed and compared the code evolution of the source and target systems to confirm the existence of the identified direct conflicts and verify if those code changes can still coexist in the target system.

To answer this research question, we have selected and manually analyzed 10 different issues. All the direct conflicts identified by MergeClear tool represent real conflicts, so we did not find any false positive. In addition to that, we also analyzed the complexity to integrate the issues with such direct conflicts. Table V shows the results for this analysis. As we can see, 60% (6 from 10) of the issues that exhibited direct conflicts could be integrated. It means that although there were code changes applied to the same artifact (method or field) in the source and target systems, they have been applied to separate parts of the code, and they are not directly related or incompatible. However, due to the difficult to automatically analyze the different semantic of such code changes, it is always necessary a manual analysis to verify the possibility of integration. Fig. 2 shows an example of a change that was considered possible to be integrated. In a same method the changes of the source and target systems were accomplished in separated "cases" of a Java switch statement. These changes are not strongly related, and they can coexist.

On the other hand, our analysis also revealed that 40% of the investigated issues (4 of 10) exhibit complex direct conflicts, which are difficult to be integrated even when applying a manual merge. Fig. 3 illustrates a change that was accomplished in the same functionality of a specific method of a class for both the source and target systems. Because these changes involve overlapping updates to the implementation of the same functionality, they are difficult to merge.

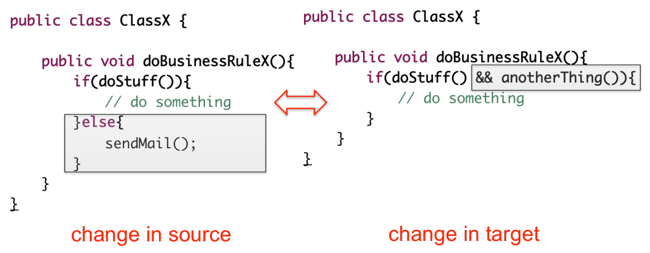
TABLE V. ANALYSIS RESULT OF THE RQ3

Analyzed Issues	Direct Conflict Possible to be Integrated	Direct Conflict Improbable to be Integrated
10	6 (60%)	4 (40%)



```
case StatusOperation.REVERT_STUDENT_EXIT:
    student.setStatus(Student.ACTIVE);
    break;


case StatusOperation.FINISH_STUDENT:
    if(student.isSpecial())
        addErrorMessage("it is not possible");
    break;

}
```

Figure 2: Direct Conflict without Integration Problem



```
public class ClassX {

    public void doBusinessRuleX(){
        if(doStuff()){
            // do something
        }else{
            sendMail();
        }
    }
}
```

```
public class ClassX {

    public void doBusinessRuleX(){
        if(doStuff() && anotherThing()){
            // do something
        }
    }
}
```

Figure 3: Direct Conflict with Integration Problem

D RQ4: Are indirect conflicts correctly identified? Do the indirect conflicts can cause behavior problems on the target system as the approach suggest?

The aim of this research question was to analyze whether the indirect conflicts collected by the MergeClear tool were correctly identified and if they could affect the behavior of the merged functionality of the changes developed for the source and target systems. In other words, we have investigated if the integration of the source and target changes that are related to indirect conflicts can cause any abnormal behavior in the system or not.

The process of manual analysis for this research question is similar to the direct conflict analysis, differing that the comparison of the evolution in the source code was made between artifacts (methods or fields) in a certain level of the call graph, and verifying if the application of this change add some error to the target system.

Table VI shows the amount of indirect conflicts identified for the evolution of the clones of the web system analyzed in this study. They were organized by the level in the call graph where they were detected. For this study we have focused on the analysis of the depth level maximum of 3. As you can see, most of indirect conflicts detected by our tool are at level 1 and 2, which justifies the analysis of indirect conflicts until the level 3.

TABLE VI. INDIRECT CONFLICTS BY LEVEL

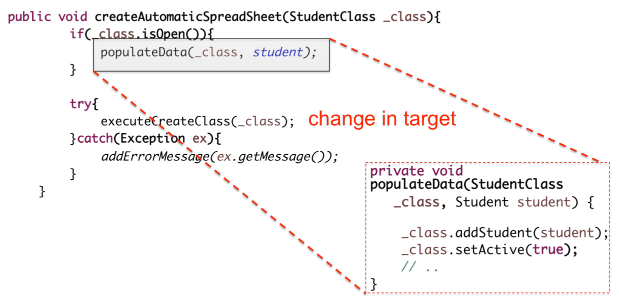
Level	Number of Indirect Conflicts	Percentage
1	338	38.72%
2	327	38.61%
3	192	22.67%

To answer RQ4, we have manually analyzed a total of 12 different issues that contains indirect conflicts. All the 12 issues analyzed represent indirect conflict defined in our approach. After that, we analyzed if those indirect conflicts could be integrated without presenting any error. We have found that 7 issues (58%) actually could add behavior problems after the integration. The analysis of the 5 remaining issues (42%) showed us that they could be integrated without causing behavior problems to the target system. This result reinforces the need to re-test issues that involves the existence of indirect merge conflicts. Table VII summarizes such results.

TABLE VII. ANALYSIS RESULT OF RQ4

Analyzed Issues	Indirect Conflict can not add behavior problem to target system	Indirect Conflict can add behavior problem to target system
12 (100%)	5 (41.66%)	7 (58.33%)

Fig, 4 shows an example of changes from one analyzed issue that generated indirect conflict in the call graph but it did not cause any behavior problem after the merge process. This example represents an extract method refactoring, which was only applied to improve the maintainability of the code. Those kinds of change in the source web system although have exhibited indirect conflicts with other code changes in the target system, they could be automatically merged without presenting any behavior problem to the target system.

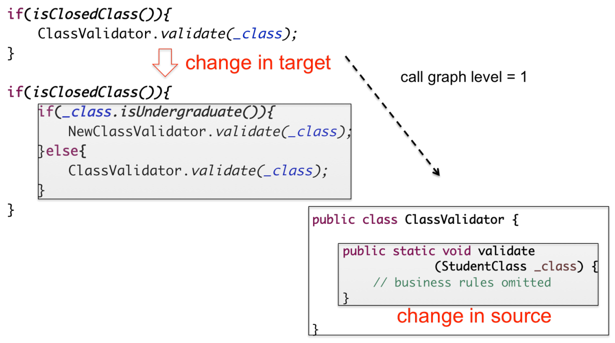


```
public void createAutomaticSpreadSheet(StudentClass _class){
        if(_class.isOpen()){
            populateData(_class, student);
        }

        try{
            executeCreateClass(_class);
        }catch(Exception ex){
            addErrorMessage(ex.getMessage());
        }
    }
```

```
private void
populateData(StudentClass
   _class, Student student) {

    _class.addStudent(student);
    _class.setActive(true);
    // ..
}
```

Figure 4: Indirect Conflict not affect target system behavior

Fig. 5 represents a change in the target system responsible to perform a specific validation for a certain kind of student. However, the original generic validation in the source system was changed during the parallel evolution of the cloned systems. Because the change in the source system did not consider the specific validation codified in target system, the merge of this code can add behavior problem to the target system.



```
if(isClosedClass()){
    ClassValidator.validate(_class);
}
```

```
if(isClosedClass()){
    if(_class.isUndergraduate()){
        NewClassValidator.validate(_class);
    }else{
        ClassValidator.validate(_class);
    }
}
```

```
public class ClassValidator {

    public static void validate
                (StudentClass _class) {
        // business rules omitted
    }
}
```

Figure 5: Indirect Conflict affect the target system behavior

IV. THREATS TO VALIDITY

Our study has focused only on the analysis of a restricted set of merge conflicts and code change operations. Other kinds of merge conflicts – such as language, semantic and test conflicts [4] – are planned to be included in our mining tool for future studies. Regarding the code change operations, our tool is currently been extended to also analyze changes on class inheritance, interface implementation, and code annotations. The results of our study are restricted to the context of the investigated web-based system, and cannot be generalized. The selection of just one clone of source and target system evolutions also represents a threat, although the selected clone contained thousands of issues. In order to expand our results, we need to replicate it for other existing systems and domains. In this direction, we plan to conduct replications of our study in the context of other existing clones from the same web-based systems presented in this paper, and to existing open-source systems from GitHub repository.

V. Related Work

Recent research work [3] [4] has investigated and proposed approaches for the analysis of code merge conflicts. Guimarães & Silva [4] propose an approach for the early and continuous detection of merging conflicts from uncommitted and committed changes in order to anticipate problems and to avoid overloading developers. They conduct an empirical study that brings evidence that the approach contributes to improve the early detection of conflicts and to avoid overloading developers in comparison with existing approaches. Brun et al [3] also conducted an empirical study considering two kinds of conflicts: (i) textual conflicts – that represent conflicts from code changes in the same artifacts; and (ii) high-order conflicts – that represent code changes that do not generate textual conflicts, but on the other hand, they cause semantic problems – compilation or test failures. Their study found for nine open-source systems that 16% of all merges present textual conflicts, and 33% of merges with no textual conflicts contain high-order conflicts. They present a quantitative and preliminary approach evaluation. In contrast, our work conducted an exploratory study of clones of a large industrial web system that quantified existing merge code conflicts and conducted a detailed analysis on the accuracy and complexity of integrating those merge conflicts using automated support.

Apel et al [12] have argued that a significant number of conflicts are ordering conflicts and show that the usage of semi-structured merge can reduce conflicts when compared to unstructured merges. Our work is consistent with their results, given the percentage of indirect and pseudo conflicts observed. However, ours is an exploratory study to better characterize and understand the kind and complexity of merge conflicts that happens in a large-scale web-based system.

Dubinsky et al [6] conducted an exploratory study of cloning in six industrial software product lines (SPLs). They found that cloning SPLs is considered a reuse approach that facilitates the independent customization of new products based on existing ones, although on the other hand, it can bring difficulties to perform maintenance and evolution activities. The merge conflict analysis approach developed presented in this paper can be used to automatically identify and possibly promote the integration of issues from one SPL clone to another. In addition to that, our work has presented concrete data related to the integration of existing cloned large-scale web systems. .

Rubin et al [7] [8] propose a framework for organizing knowledge related to the development, maintenance and merge-refactoring of product lines realized via cloning. They organize such framework in terms of a set of clone management operators. They also performed a detailed analysis of development issues of industrial SPL companies in terms of these operators. Indeed, the web-based system investigated in our work can be seen as a cloned product lines that is evolving independently to accommodate new variabilities. In this paper, we have proposed a merge conflict analysis approach to understand and promote the integration of development issues that can also be used in the context of cloned SPLs. While Rubin et al [7] [8] propose a general and language independence approach, we restrict our approach to system implemented in the Java language, which allowed achieve more concrete results in our analysis.

VI. Conclusion

This paper presented an exploratory study of characterization of merge conflicts in the context of cloned web-based information systems. In our study, we have found: (i) a considerable number of indirect merge conflicts compared to direct and pseudo conflicts when merging cloned web-based-systems; and (ii) the feasibility to use merge analysis approaches to integrate tasks from one cloned system to another one considering the kinds of merge conflicts analyzed in our study – direct, indirect and pseudo. Finally, we also found that the integration of issues from source to target systems also requires the computation and resolution of dependent issues that were previously developed in the source system. As a future work, we plan to replicate our study to other cloned web-based information systems from our institution and from other companies in order to have a better understanding of the cloning impact for this domain of applications. In these new studies, we are also including other categories of conflicts and code change operations. It is also necessary to improve the dependency detection algorithm to address the integration of dependent issues.

References

[1] T. Mens. A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering (TSE 2002), Vol. 28, 5.

[2] J. Wokla, B. Ryder, F. Tip, X. Ren. Safe-Commit Analysis to Facilitate Team Software Development. Proceedings of ICSE 2009.

[3] Y. Brun, et al. Early Detection of Collaboration Conflicts and Risks, IEEE Trans. on Software Engineering, vol. 39, no. 10, pp. 1358-1375.

[4] Guimarães, M. L. & Silva, A. R, 2012. Improving early detection of software merge conflicts. InProceedings of the 2012 International Conference on Software Engineering (ICSE 2012). IEEE Press, Piscataway, NJ, USA, 342-35

[5] T. Zimmermann, "Mining Workspace Updates in CVS," Proc. Fourth Int'l Workshop Mining Software Repositories, May 2007.

[6] Y. Dubinsky, et al. An Exploratory Study of Cloning in Industrial Software Product Lines. Proceedings of CSMR 2013.

[7] J.Rubin,et al. 2012. Managing Forked Product Variants. Proceedings of the Software Product Line Conference (SPLC 2012). Salvador. Brazil

[8] J. Rubin, K. Czarnecki, M. Chechik. Managing cloned variants: a framework and experience. Proceedings of SPLC 2013: 101-110.

[9] G. Lima, et al. A Delta Oriented Approach to the Evolution and Reconciliation of Enterprise Software Products Lines. ICEIS (1) 2013: 255-263

[10] MergeClear. A tool for merge cloned systems http://github.com/jadsonjs/MergeClear (last visited on August 27, 2015)

[11] Product Line Hall of Fame. http://splc.net/fame.html (last visited on August 27, 2015)

[12] S. Apel, J. Liebig, B. Brandl, C. Lengauer, C. Kästner: Semistructured merge: rethinking merge in revision control systems. Proceedings of FSE 2011

[13] MergeClear Wiki. Welcome to the MergeClear wiki. http://github.com/jadsonjs/MergeClear/wiki (last visited on August 27, 2015)