Tracking the Ripple Impact of Code Changes through Version Control Systems

Ali Irfan, Arif Ur Rahman
Department of Computer Science
Bahria University, Islamabad, Pakistan
{aliirfan84,badwanpk}@gmail.com

***Abstract*—Continuous updates and modifications are integral part of software development and maintenance process. With the advancements in technology, geographically distributed teams can contribute to code of a project shared with the help of integrated development environments (IDE) like Visual Studio and version control systems (VCS) like VSTS. In addition to managing shared code repository, VCS helps in keeping track of changes made in the source code. However, sometimes developers may detect the ripple impact of changes made in a specific code file. For this purpose, developers have to obtain the specific version of source code file from VCS and then use IDE to track the references of modified methods. This paper proposes an approach that enables VCSs to maintain a log of files and methods that have been affected by changes in a specific method within a project. This will assist developers in reviewing the impact of a specific change prior to merging the code files into their working directory. The proposed technique can be linked with any VCS like GIT, VSTS or Subversion (SVN) in the form of extensions. The effectiveness of having this capability withing the VCS is evaluated through a survey pertaining relevant questions. Survey result yields that having visibility of the source code impacts within the VCS will ease development and review process.**

***Index Terms*—Version Control System, Integrated Development Environment, Visual Studio Team Services, Subversion, Change Impact Analysis**

I. Introduction

Constant evaluation and changes in source code are integral part of the software development life cycle. The process to identify the impact of a certain change to the related source code is generally referred to as change impact analysis (CIA) [1]–[6]. Working with object-oriented programming (OOP), logic is encapsulated and exposed in the form of methods that may be referenced and consumed by other programs or processes. Most of the modern Integrated Development Environments (IDEs) like Visual Studio and Eclipse provide rich IntelliSense using which programmers can easily view references against a given method. This helps programmers in analyzing the impact of change in a given method on other areas of the application. Modern software development activities require multiple programmers to contribute to a single project. Moreover, the programmers may be geographically spread across multiple locations. In such a scenario where multiple programmers have to collaborate and they are geographically spread across multiple locations, the need to properly maintain a shared code base and tack change history arises. Typically, programmers use code repositories and version control systems (VCS) to achieve the goals in a systematic manner [7], [8]. VCS is a software application that helps in collaborative software development. Typical examples of VCS include Visual Studio Team Services (VSTS), Git, and Subversion. A VCS keeps track of changes in code files along with related metadata but there are some limitations associated [2]. Moreover, unlike IDEs, VCSs do not keep track of impediments that may occur in response to modifications in a method.

An important aspect of the software development activity is to identify the impact of a change across other parts of the source code, generally referred to as change impact analysis (CIA). There have been a lot of static and dynamic analysis techniques to identify the ripple effects of change before and during the development phase [9]. IDEs like Visual Studio provide static code analysis, identify method references and assist programmers to view the impact of new alterations. After completion of the task, developers merge code into the VCS so the changes or updated code files are available to rest of concerned team members. VCS provides the facility to trace back changes made in a specific file.

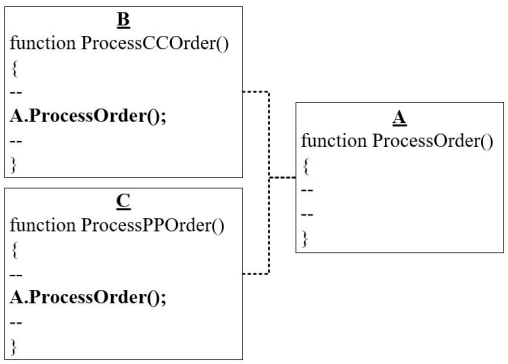


Fig. 1. Method Dependency

However, there is no way to view possible impact areas caused by the checked-in or merged file. The changes in a method can be divided into two categories i.e. change in method signatures and change in the method definition.

978-1-7281-6625-4/19/$31.00 ©2019 IEEE
DOI 10.1109/FIT47737.2019.00056

For illustration, consider the following scenario: there are three class files, namely A, B, and C. There is a method called `ProcessOrder()` defined in file A whereas file B and C consume this method to complete a business transaction as presented in figure 1. For any change in method signatures like the addition of argument or change in the return type, modern IDEs will trigger an error in file B and C so developers are required to make necessary adjustments. However, if there is a change in the method definition, IDEs will not raise any warning or notification to the programmers and they have to manually look into the references where `A.ProcessOrder()` is consumed.

Once file A is merged into the VCS, there is no way to trace back the ripple effects of changes to the other areas of the application (i.e. file B and C). In order to analyze the impact of change, programmers have to get the latest file (i.e. A) in their local code repository, compare which part or methods of the file are changed and manually view the references.

This paper proposes an approach to keep track of the ripple impacts of a change in the specific file after it has been merged into the VCS. The proposed logic can be implemented as a plugin or extension with tools like VSTS. The rest of this paper is organized as follows: Section II presents the background and related work. Section III describes the proposed solution and the final section presents some conclusions.

II. Background

In general change impact analysis in different software artefacts is an exciting research topic. In order to identify logical and performance related ripple effects during maintenance of a large-scale software development system, a mathematical model was proposed to calculate the complexity of modifications that helps to determine the effort required by programmers to make necessary changes [3]. However, the paper does not include any algorithm or implementation details. The importance of identifying the impacts in the source code at various granularity levels have been discussed in detail [4]. Most of the IDEs and extensions provide the functionality to check method dependency at a single granularity level. They proposed a technique called Augur and built an extension for Visual Studio to detect both data and control dependencies with change impact query language.

Version Control Systems (VCS) play vital role in detecting potential impacts that might have implicit dependencies especially in software product lines (SPL) [5]. The study applied delta-oriented implementation technique to minimize coupling and enhance re-usability. The proposed approach CIAHelper takes deltas and models as input and generate dependency graphs for delta-programming based SPLs. Function point analysis has always been the most favourite technique for effort estimation. In continuation, it is proposed to use CIA technique in combination with static impact analysis (SIA) and dynamic impact analysis DIA for better visualization of required effort [10].

Transaction dependency graph model (TDGs) are useful to determine the long-term impact of changes and notify programmers about the possible risk of making alternations [11]. To determine the impact of changes employing both SIA and DIA techniques, a prototype CIAT is developed [9]. The study argued that Use Case Map (UCM) is not much efficient to conduct static impact analysis due to missing trace-ability between functionality requirement and actual source code. Similarly, path impact and influence mechanism techniques used for dynamic impact analysis are not efficient and time-consuming. VCS can help to identify bad chunks of code by analysing historical data. The proposed approach extracts the history for code changes from VCS and then apply different code smell detection techniques like shotgun surgery detection, feature envy detection, blob detection, and parallel inheritance. Since VCS does not provide a facility to automatically provide the impact of method/class changes, the implementation depends on manual detection and correlating the impact of method/class changes to the other parts of the source code.

This work helps to identify and track the ripple impact of ongoing development changes to the other areas of application. However, they are less likely to help visualize impact and trace-ability after the code has been merged into the version control system. In order to help developers and software support team, its important to provide traceability graph against change in one file or method to all the other areas where this functionality is invoked. To strengthen this hypothesis, a survey was conducted in which participants were asked various questions regarding the current practices they are following to detect the impact of checked-in code into the VCS. A questioner was distributed through Google Forms among software developers of the various organizations. The participants were inquired about the availability and usefulness of the integrated impact detection feature into the VCS.

Only those respondents were selected who are directly involved in the development activity with object-oriented programming (OOP) paradigm. The restriction was put to enhance the confidence of survey results. It was made sure that respondents are using IDE for writing code along with a VCS to manage shared code repository across different team members.

The answer to the following questions helped to clarify and strengthen the need for tracking the impact of source code changes visibility directly from within the VCS.

Q1. Does the version control system in your development environment helps to track the ripple impact of a modified method without fetching the latest copy of source code into your local working directory?

Q2. Do you have to obtain latest copy of the source code into your working directory to review and track ripple impacts?

Q3. In your opinion, should there be any built-in capability in VCS to track the impact of source code changes?

Q4. Does tracing capability against changes merged into VCS help identify possible vulnerabilities in timely fashion and gear up review process?

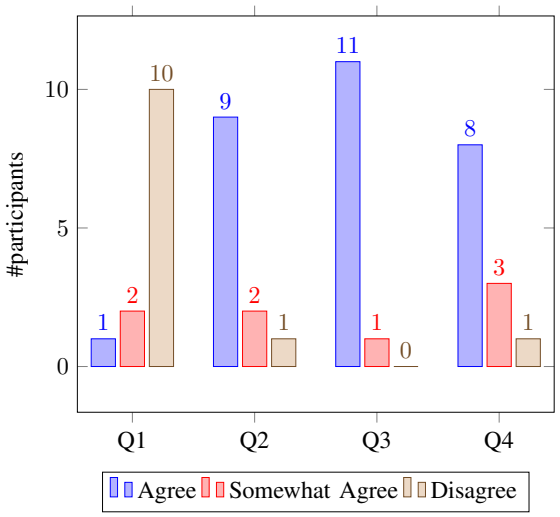


Fig. 2. Survey Results

Team Foundation Server (TFS)/Visual Studio Team Services (VSTS) is used by 50% of the respondents, GIT is used by 33% while the rest are using visual source safe as VCS tool. 91% of the respondents agreed that there should be an in-built facility within VCS to track the impact of checked-in source code; 82% agreed that this will help to identify the origin of possible bugs in a timely fashion. It is evident from the findings that developers used to obtain modified files from the VCS into their local working directory and then identify the impact of modified code, specially modified methods, using `Find References` feature of IDEs. Respondents highlighted the need for having an in-built impact detection feature in VCS. Moreover, they are of the opinion that it will help to improve tracing and improve the overall efficiency of the development team.

III. Proposed Solution

This study proposes a mechanism to keep track of change dependencies as soon as a file is checked-in into the VCS. Most of the VCSs keep copies of code files upon every merge. Its is proposed to extend the default file merge functionality by extracting and supplying direct and indirect dependencies of changed methods.

Figure 3 depicts an abstract level view of how Change Dependency Register (CDR) can be integrated with IDE and VCS. It is proposed to introduce two additional capabilities namely dependency identifier (DI) and dependency repository (DR). The primary job of DI is to extract the references of all modified functions against all files that are going to be merged into the VCS. For this purpose, DI can utilize IDEs built-in capability with the help of programming extensions. For example, in the case of Visual Studio, DI an extract functional dependency by implementing or extending functionality against `Find All References` command. Once this information is extracted, the next step is to log this information along with the merged file(s). For this purpose, there is a requirement to store the 'references information' against merged files, ideally in a similar fashion as the underlying VCS repository. For instance, Microsoft Team Foundation Server (TFS) uses SQL Server to maintain change log; we can simply add another entity in the underlying database to store references and reference this back to the modified records in file repository log. However, this information can be stored separately in the form of hierarchical structure like XML file and later can be extracted through pointers.

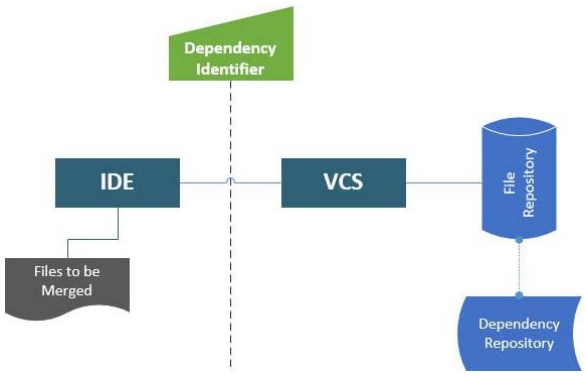


Fig. 3. CDR Implementation Pattern

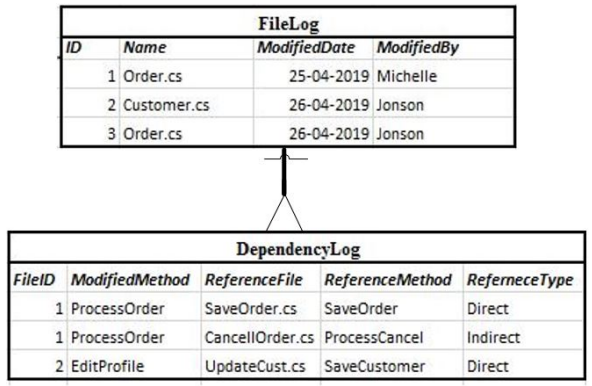
Table I lists the minimum set of attributes that are to be maintained by DR either in the form of a relation or XML structure.

TABLE I
DR Repository Schema

Attribute	Description
FileID	Reference of the file that is merged into the VCS; this can be Foreign Key of the table that keeps a log of changed files.
ModifiedMethod	Name of the method modified in checked in file
ReferenceFile	Name of the file containing functionality that might have an impact due to a recent update in associated method
ReferenceMethod	The actual name of the method in ReferenceFile that might have an impact
ReferenceType	Direct: ModifiedMethod is directly referenced/ consumed by RefernceMethod Indirect: ReferenceMethod is consuming a method that in-turn is consuming ModifiedMethod

For illustration purpose, suppose a developer has modified the `ProcessOrder()` method in code file named `Order.cs`. Moreover, the `ProcessOrder()` method is consumed by two other methods, namely `SaveOrder()` and `ProcessCancel()` in code files `SaveOrder.cs` and `CancelOrder.cs` respectively. Since all of these files belong to the same project, its pretty straightforward to extract reference information against the `ProcessOrder()` method. Once `Order.cs` file is merged into the VCS, dependency information will be extracted by DI and saved into VCS repository for future use. Figure 4 provides an overview

of how this information will be maintained by DR within the VCS repository.



FileLog

ID	Name	ModifiedDate	ModifiedBy
1	Order.cs	25-04-2019	Michelle
2	Customer.cs	26-04-2019	Jonson
3	Order.cs	26-04-2019	Jonson

DependencyLog

FileID	ModifiedMethod	ReferenceFile	ReferenceMethod	RefernceType
1	ProcessOrder	SaveOrder.cs	SaveOrder	Direct
1	ProcessOrder	CancelIOrder.cs	ProcessCancel	Indirect
2	EditProfile	UpdateCust.cs	SaveCustomer	Direct

Fig. 4. CDR Implementation Pattern

There will be a one-many relationship between `FileLog` and `DependencyLog` as there can be many dependencies against modification in a single file. There can be changes to one or multiple methods in a file having multiple direct or indirect references to dependent methods across single or multiple files. Since dependency relationship log is maintained by DR unit of the VCS, this information can later be extracted and displayed to the users upon request.

Algorithm 1 Log Modified References in DR

```
1: Input: List of Files Checked-in
2: Output: D_f List of Dependent Files

3: procedure LogDependency(F_i)
4:     for each f in F_i do
5:         M_i = Extract Modified Methods in f
6:         for each m in M_i do
7:             D_f = Direct Dependencies of m
8:             LogInDR(D_f) //Log entry in Repository
9:             LogDependency(D_f) //Check indirect refs
10:        end for
11:    end for
12: end procedure
```

Algorithm 1 presents the pseudocode to log method references in DR. The `LogDependency` method expects a parameter containing the list of modified files along with respective modified methods. For example, lets assume there is a separate list of modified methods against each file `f` in F_i. Each file is iterated to identify the list of modified methods M_i. In the subsequent step, direct method dependencies D_f are identified against each modified method `m` and log into the dependency repository (DR). In order to find indirect references, the same routine is invoked recursively for each D_f. This process will continue until all the files in the initial set of F_i have been processed. Once we have desired data in DR, any integrated routine or plugin can consume this data to notify VCS users about the impacts of certain change.

IV. Conclusions and Future Work

Tracking the origin and impact of a changes across other parts of the source code is very critical especially after the code has been merged into the version control system. The proposed technique encourages to maintain a log of affected files and methods along with the original change within the VCS. This way developers will be capable to review the ripple-effects a change directly from within the version control system. Keeping log of impacted files and methods can further be utilized for various purposes including but not limited to bad smell detection. In the future, we will be implementing the proposed technique using one of the VCS as an extension and distribute the plugin among the software developers who participated in survey.

References

[1] Bohner, S and Arnold, R, "Software Change Impact Analysis," in *An Introduction to Software Change Impact Analysis*. Los Alamitos, CA: IEEE Computer Society Press, 1996, pp. 1–28.

[2] R. Majumdar, R. Jain, S. Barthwal, and C. Choudhary, "Source code management using version control system," in *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Sep. 2017, pp. 278–281.

[3] S. S. Yau, J. S. Collofello, and T. MacGregor, "Ripple effect analysis of software maintenance," in *The IEEE Computer Society's Second International Computer Software and Applications Conference, 1978. COMPSAC '78.*, Nov 1978, pp. 60–65.

[4] T. Sharma and G. Suryanarayana, "Augur: Incorporating hidden dependencies and variable granularity in change impact analysis," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Oct 2016, pp. 73–78.

[5] M. Hamza, R. J. Walker, and M. Elaasar, "CIAhelper: Towards Change Impact Analysis in Delta-oriented Software Product Lines," in *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 1*, ser. SPLC '18. New York, NY, USA: ACM, 2018, pp. 31–42.

[6] T. Rolfsnes, L. Moonen, and D. Binkley, "Predicting relevance of change recommendations," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct 2017, pp. 694–705.

[7] R. Glassey, "Adopting git/github within teaching: A survey of tool support," in *Proceedings of the ACM Conference on Global Computing Education*, ser. CompEd '19. New York, NY, USA: ACM, 2019, pp. 143–149.

[8] V. Kovalenko, F. Palomba, and A. Bacchelli, "Mining file histories: Should we consider branches?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 202–213.

[9] S. Basri, N. Kama, R. Ibrahim, and S. A. Ismail, "A Change Impact Analysis Tool for Software Development Phase," *International Journal of Software Engineering and its Applications*, vol. 9, no. 9, pp. 245–256, 2015.

[10] J. Shah, N. Kama, and S. A. Ismail, "An Empirical Study with Function Point Analysis for Software Development Phase Method," in *Proceedings of the 7th International Conference on Software and Information Engineering*, ser. ICSIE '18. New York, NY, USA: ACM, 2018, pp. 7–11.

[11] K. S. Herzig, "Capturing the long-term impact of changes," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2, May 2010, pp. 393–396.