CRISTIANO MACHADO CESÁRIO

AWARENESS OVER DISTRIBUTED VERSION CONTROL SYSTEMS

Thesis presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Topic Area: Software Engineering.

Advisor: Prof. D.Sc. Leonardo Gresta Paulino Murta

Niterói

2014

Ficha Catalográfica – Esta página deve ser removida na versão a ser entregue para a banca, mas deve ser reinserida na versão final, com a ficha catalográfica fornecida pela biblioteca. Informações sobre este processo devem ser obtidas na secretaria da pós-graduação.

CRISTIANO MACHADO CESÁRIO

AWARENESS OVER DISTRIBUTED VERSION CONTROL SYSTEMS

Thesis presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Topic Area: Software Engineering.

Approved on <MES> <ANO>.

APPROVED BY

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. D.Sc. Leonardo Gresta Paulino Murta – Advisor

UFF

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. D.Sc. <NOME DO AVALIADOR>

<INSTITUIÇÃO DO AVALIADOR>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. D.Sc. <NOME DO AVALIADOR>

<INSTITUIÇÃO DO AVALIADOR>

Niterói

2014

dedicatória(s): Folha onde o autor presta homenagem ou dedica seu trabalho.

**acknowledgments**

dedicatória(s): Folha onde o autor presta homenagem ou dedica seu trabalho.

dedicatória(s): Folha onde o autor presta homenagem ou dedica seu trabalho.

**Resumo**

O desenvolvimento de software utilizando Sistemas de controle de versão distribuídos tem se tornado cada vez mais frequente recentemente. Tais sistemas trazem mais flexibilidade, mas também trazem uma maior complexidade para administrar e monitorar os múltiplos repositórios existentes, assim como a proliferação de vários ramos. Neste trabalho, propomos o DyeVC, uma abordagem extensível para auxiliar desenvolvedores e administradores de repositórios a identificar dependências entre os repositórios distribuídos, como forma de ajudar a entender o que acontece ao redor do repositório de alguém e descobrir as relações entre os repositórios existentes.

**Palavras-chave: Gerência de configuração, Percepção de espaços de trabalho, Controle de versão distribuído, Evolução de repositórios**

**Abstract**

Software development using distributed version control systems has become more frequent recently. Such systems bring more flexibility, but also bring greater complexity to administer and monitor the multiple existing repositories as well as the proliferation of several branches. In this work we propose DyeVC, an extensible tool to assist developers and repository administrators in identifying dependencies amongst the distributed repositories in order to help to understand what is going on around one’s repository and depict the relations between the existing repositories.

**Keywords: Configuration management, Workspace awareness, Distributed version control, Repository evolution**

**List OF FIGURES**

[Figure 1 - A development scenario involving some developers 17](#_Toc393356493)

[Figure 2 - How DyeVC gathers information 21](#_Toc393356494)

[Figure 3 - Model used to store topology data 22](#_Toc393356495)

[Figure 4 - DyeVC showing notifications in the notification area 23](#_Toc393356496)

[Figure 5 - Topology view for a given project 24](#_Toc393356497)

[Figure 6 - DyeVC Main Screen 24](#_Toc393356498)

[Figure 7 - Commit history for a given project 26](#_Toc393356499)

[Figure 8 - Collapsed commit history 27](#_Toc393356500)

[Figure 9 - Updating repositories in the topology 28](#_Toc393356501)

[Figure 10 - Topology view showing first monitored repository 33](#_Toc393356502)

[Figure 11 - aakoch’s commit history showing commits pending to be pushed 33](#_Toc393356503)

[Figure 12 - Topology view showing the three monitored repositories 33](#_Toc393356504)

[Figure 13 - Adam’s tracked branches 34](#_Toc393356505)

[Figure 14 - Aakoch’s commit history 35](#_Toc393356506)

[Figure 15 - Jeresig’s tracked branches 35](#_Toc393356507)

**List OF TABLES**

[Table 1 - Possible States of a Repository 25](#_Toc393356508)

[Table 2 - Status of a local repository regarding a remote one, based on the existence of non-replicated commits 25](#_Toc393356509)

[Table 3 - Existing commits in each repository 25](#_Toc393356510)

[Table 4 - Status of each repository based on known remote repositories 26](#_Toc393356511)

[Table 5 - Time (in seconds) spent by DyeVC in several operations for repositories with different histories and sizes 37](#_Toc393356512)

[Table 6 - Pearson coefficient between measured operations and repository metrics 37](#_Toc393356513)

**List OF ACRONYMS AND ABBREVIATIONS**

Elemento opcional, que consiste na relação alfabética das abreviaturas e siglas utilizadas no texto, seguidas das palavras ou expressões correspondentes grafadas por extenso. Recomenda-se a elaboração de lista própria para cada tipo.

**List of symbols**

Elemento opcional, que deve ser elaborado de acordo com a ordem apresentada no texto, com o devido significado.

TABLE OF CONTENTS

[Chapter 1 – Awareness and Distributed Version Control Systems 14](#_Toc394512840)

[1.1 Introduction 14](#_Toc394512841)

[1.2 Distributed Version Control Systems 15](#_Toc394512842)

[1.3 Related Work 16](#_Toc394512843)

[1.4 Final Considerations 18](#_Toc394512844)

[Bibliography 19](#_Toc394512845)

# – Awareness and Distributed Version Control Systems

## Introduction

SCM is part of the Software Engineering and it was born in the 70s (ESTUBLIER, 2000). According to (MURTA, 2006), under the development perspective, SCM is divided into three main systems: Change Management, Version Control and Release Management. **Change Management** is in charge of systematically controlling the configuration, storing and reporting the information produced along the change requests. **Version Control** deals with the identification and evolution of configuration items – CI. Finally, **Release Management** automates the process of building executables from the source code and releasing them in production.

Version Control Systems (VCSs) date back to the 70s, when the first VCS, called Source Code Control System – SCCS, emerged (ROCHKIND, 1975). VCSs were the first SCM systems to emerge and, since then, they have evolved substantially, from the local access needed by SCCS, to a client-server architecture, and more recently to a distributed architecture. Besides their original scope, they have also been used as a data source to mine data related to software development, because they have the knowledge of all CIs and how they evolved over the time (what was changed, why they were changed, when they were changed and who performed the change). Moreover, as VCSs are the single point where every IC resides, they might be used not only to know what happened in the past, but also to be aware of what is happening in the present and predict what might happen in the future.

Awareness is defined by (DOURISH; BELLOTTI, 1992) as “an understanding of the activities of others to provide a context for one’s own activities”. The information needed to provide awareness depends on what people need to be aware of. In the context of this work, which focus on the awareness of changes occurring in DVCS, all the information needed is generally available within the repository, such as which files have changed, what changes were applied, who applied the changes, when the changes occurred and why the changes were applied.

This chapter presents some basic concepts related to DVCS and some approaches related to providing awareness combined with VCS. Section 1.2 discusses central concepts regarding how DVCS work. Section 1.3 presents related work. Finally, Section 1.4 presents the final considerations of this chapter.

## Distributed Version Control Systems

CVCS relies on a centralized repository, stored on a server (Figure 1.a). When one wants to work on any CI, a *checkout* is performed, that copies a specific version of the artifacts from the repository to a workspace where changes can be applied. Later, after applying the changes, a *commit* (also known as *check-in)* is performed, that sends all the changes back to the repository. Updates made by other people can be brought to the workspace at any desired time, by performing an *update*. Updates are automatically merged into workspace, or in the case of physical conflicts (same line changed locally and remotely), the developer might have to handle the merge manually.

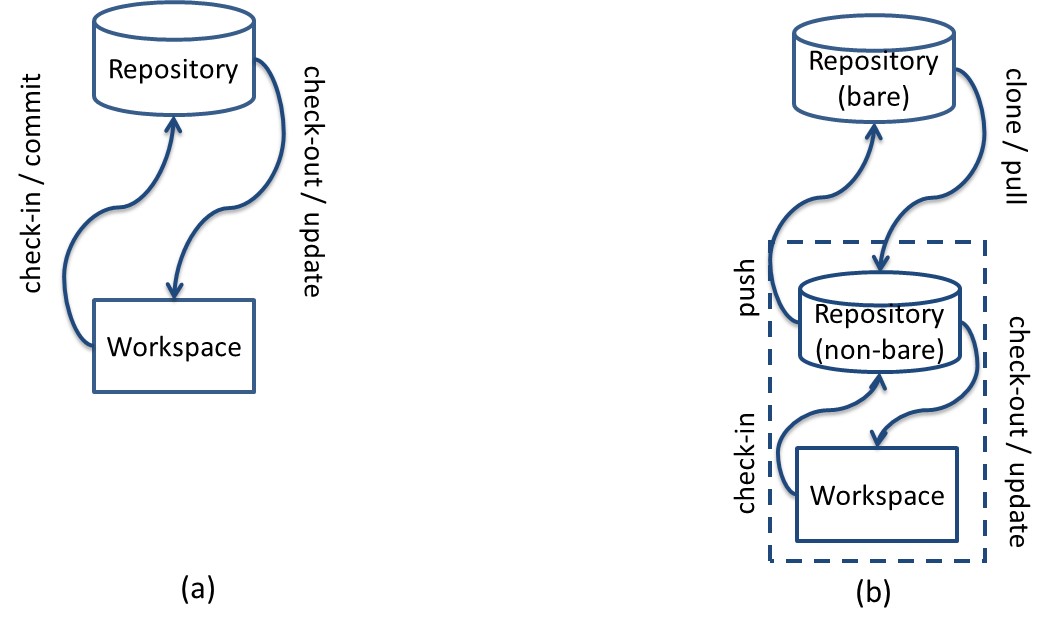


Figure 1 – CVCS (a) versus DVCS (b)

DVCS, on the other hand, does not rely on a centralized repository. It uses an architecture where the entire repository is distributed and it exists in every machine where one wants to work with it. The changes continue to take place on a workspace, but this time there is a local copy of the repository (Figure 1.b). The main operations (*checkout, commit* and *update*)continue to exist, but are performed locally, allowing one to work offline, committing work whenever necessary. However, another set of commands arise, that allows sending and receiving changes between different repositories. Initially, a *clone* is performed, which copies a repository to a specified location. The copy can be done from a repository located on a partner or on a server. There is no concept of a *central* repository, but one of the distributed repositories can act as a central one by having, for example, a strict policy regarding who might send changes to it. Repositories might either have an associated workspace or not. A repository located on a server, where local changes are not expected to occur, does not need to have a workspace associated, and we refer to it as a *bare* repository. Changes can be sent from a local repository to a remote repository by invoking a *push* command. Changes can be received from remote repositories and applied immediately to the workspace, leading to merges / conflicts, or they can stay at the local repository to be applied later. A *pull* command brings the changes and applies them to the workspace. A *pull* can be broken into two subcommands: a *fetch* command, that only transfers the changes to the local repository, without applying them to the workspace, and a *merge* command, which applies the changes to the workspace.

## Related Work

According to (DIEHL, 2007) software visualization can be separated into three aspects: structure, behavior, and evolution. DyeVC relates primarily with the evolution aspect, more specifically with studies that aim at improving the awareness of developers that work with distributed software development. A recent work by (STEINMACHER; CHAVES; GEROSA, 2012) presents a systematic review of awareness studies and classify them according to the Awareness Framework (GUTWIN; GREENBERG; ROSEMAN, 1996) and according to the 3C Collaboration Model (FUKS *et al.*, 2007). The classification is not exclusive, i.e., a given tool can present elements of different awareness types. Following (GUTWIN; GREENBERG; ROSEMAN, 1996), DyeVC can be classified as a “Workspace Awareness” approach and according to (FUKS *et al.*, 2007), DyeVC fits into the “Coordination” and “Cooperation” categories.

This section describes some approaches related to awareness or visualization of information stored in VCSs. We used a criterion similar to snowballing sample (GOODMAN, 1961) to select the approaches, starting with a finite individual population as a seed and looking for these approaches’ citations and at approaches that cited them. Our initial seed was based on the referenced papers analyzed by (STEINMACHER; CHAVES; GEROSA, 2012). We also searched at the main academic digital libraries (ACM, IEEE, SpringerLink e ScienceDirect) and at the industry, using the keywords “revision”, “souce code”, “software configuration”, “source control”, “version control”, “application” and “system”, combined with “awareness” and “visualization”. We filtered the results found to get only studies that used any VCS. The resulting studies were divided into three groups. The first group includes tools that notify commit activities. The second group comprises of approaches that not only give the developer awareness of concurrent changes, but also inform them if conflicts were detected. Finally, the third group includes approaches that visualize repository information in a linear way.

The first group contains tools such as *SVN Notifier*[[1]](#footnote-1), *SCM Notifier*[[2]](#footnote-2), *Commit Monitor*[[3]](#footnote-3), *SVN Radar*[[4]](#footnote-4), *Hg Commit Monitor*[[5]](#footnote-5)and Elvin (FITZPATRICK; MARSHALL; PHILLIPS, 2006)*.* All of them focus on avoiding conflicts by increasing the developer’s perception of concurrent work. In addition, they are generally simple tools without extensive research or published work (except for (FITZPATRICK; MARSHALL; PHILLIPS, 2006)). They fail to identify related repositories and do not provide information in different levels of details, such as status, branches, and commits. DyeVC provides these different levels of details, as shown in Section 0.

The second group includes tools such as *Palantir*, (SARMA; VAN DER HOEK, 2002), *CollabVS* (DEWAN; HEGDE, 2007), *Crystal* (BRUN *et al.*, 2011), *Safe-Commit* (WLOKA *et al.*, 2009), *Lighthouse* (DA SILVA *et al.*, 2006), *FASTDash* (BIEHL *et al.*, 2007), and *WeCode* (GUIMARÃES; SILVA, 2012). Among these studies, only *Crystal* and *FASTDash* work with DVCSs. *Crystal* detects physical, syntactic, and semantic conflicts in Git repositories (provided that the user informs the compile and test commands), but does not deal with repositories that pull updates from more than one peer*.* *FASTDash* does not detect conflicts directly, as the previous cited studies, but provides awareness of potential conflicts, such as two programmers editing the same region of the same source file, in repositories stored in Team Foundation Server[[6]](#footnote-6). According to the authors, *FASTDash* was designed for project teams of 3-8 developers, which makes it inappropriate to be used in large projects. Although DyeVC primary focus is not to detect conflicts, it can be seen as a supporting infrastructure that can be combined with such approaches to allow conflicts and metrics analysis over DVCS.

In the third group, the visualization can be done with different focuses, such as program structures (COLLBERG *et al.*, 2003), classes (LANZA, 2001), lines (VOINEA; TELEA; VAN WIJK, 2005), authors (GILBERT; KARAHALIOS, 2006), and branch history (ELSEN, 2013; HOZUMI, [N.a.]; KOIKE; CHU, 1997; PRESTON-WERNER, [N.a.]). The latter is the focus of DyeVC’s Commit History visualization.

Most of these works were applied only to CVCSs. The only exception found were (ELSEN, 2013; HOZUMI, [N.a.]) and (PRESTON-WERNER, [N.a.]), which work with Git repositories, but look only at a local repository, not showing, for example, where a given commit can be found.

Regarding the topology view in DyeVC, we could not find any similar work.

## Final Considerations

The first group contains tools such as *SVN Notifier*[[7]](#footnote-7), *SCM Notifier*[[8]](#footnote-8), *Commit Monitor*[[9]](#footnote-9), *SVN Radar*[[10]](#footnote-10), *Hg Commit Monitor*[[11]](#footnote-11)and Elvin (FITZPATRICK; MARSHALL; PHILLIPS, 2006)*.* All of them focus on avoiding conflicts by increasing the developer’s perception of concurrent work. In addition, they are generally simple tools without extensive research or published work (except for (FITZPATRICK; MARSHALL; PHILLIPS, 2006)). They fail to identify related repositories and do not provide information in different levels of details, such as status, branches, and commits. DyeVC provides these different levels of details, as shown in Section 0.

The second group includes tools such as *Palantir*, (SARMA; VAN DER HOEK, 2002), *CollabVS* (DEWAN; HEGDE, 2007), *Crystal* (BRUN *et al.*, 2011), *Safe-Commit* (WLOKA *et al.*, 2009), *Lighthouse* (DA SILVA *et al.*, 2006), *FASTDash* (BIEHL *et al.*, 2007), and *WeCode* (GUIMARÃES; SILVA, 2012). Among these studies, only *Crystal* and *FASTDash* work with DVCSs. *Crystal* detects physical, syntactic, and semantic conflicts in Git repositories (provided that the user informs the compile and test commands), but does not deal with repositories that pull updates from more than one peer*.* *FASTDash* does not detect conflicts directly, as the previous cited studies, but provides awareness of potential conflicts, such as two programmers editing the same region of the same source file, in repositories stored in Team Foundation Server[[12]](#footnote-12). According to the authors, *FASTDash* was designed for project teams of 3-8 developers, which makes it inappropriate to be used in large projects. Although DyeVC primary focus is not to detect conflicts, it can be seen as a supporting infrastructure that can be combined with such approaches to allow conflicts and metrics analysis over DVCS.

In the third group, the visualization can be done with different focuses, such as program structures (COLLBERG *et al.*, 2003), classes (LANZA, 2001), lines (VOINEA; TELEA; VAN WIJK, 2005), authors (GILBERT; KARAHALIOS, 2006), and branch history (ELSEN, 2013; HOZUMI, [N.a.]; KOIKE; CHU, 1997; PRESTON-WERNER, [N.a.]). The latter is the focus of DyeVC’s Commit History visualization.

Most of these works were applied only to CVCSs. The only exception found were (ELSEN, 2013; HOZUMI, [N.a.]) and (PRESTON-WERNER, [N.a.]), which work with Git repositories, but look only at a local repository, not showing, for example, where a given commit can be found.

Regarding the topology view in DyeVC, we could not find any similar work.

Bibliography

BIEHL, J. T.; CZERWINSKI, M.; SMITH, G.; ROBERTSON, G. G. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. CHI ’07, 2007, New York, NY, USA: ACM, 2007. p. 1313–1322. . Accessed: 3 may 2014.

BRUN, Y.; HOLMES, R.; ERNST, M. D.; NOTKIN, D. Proactive detection of collaboration conflicts. ESEC/FSE ’11, sep. 2011, New York, NY, USA: ACM, sep. 2011. p. 168–178.

COLLBERG, C.; KOBOUROV, S.; NAGRA, J.; PITTS, J.; WAMPLER, K. A System for Graph-based Visualization of the Evolution of Software. SoftVis ’03, 2003, New York, NY, USA: ACM, 2003. p. 77–ff. . Accessed: 4 may 2014.

DA SILVA, I. A.; CHEN, P. H.; VAN DER WESTHUIZEN, C.; RIPLEY, R. M.; VAN DER HOEK, A. Lighthouse: coordination through emerging design. eclipse ’06, oct. 2006, New York, NY, USA: ACM, oct. 2006. p. 11–15.

DEWAN, P.; HEGDE, R. Semi-synchronous conflict detection and resolution in asynchronous software development. ECSCW 2007, sep. 2007, Limerick, Ireland: Springer London, sep. 2007. p. 159–178.

DIEHL, S. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. 2007 edition ed. Berlin ; New York: Springer, 2007.

DOURISH, P.; BELLOTTI, V. Awareness and Coordination in Shared Workspaces. CSCW ’92, 1992, New York, NY, USA: ACM, 1992. p. 107–114. . Accessed: 5 mar. 2014.

ELSEN, S. VisGi: Visualizing Git branches. In: 2013 FIRST IEEE WORKING CONFERENCE ON SOFTWARE VISUALIZATION (VISSOFT), sep. 2013, Eindhoven, Netherlands: IEEE, sep. 2013. p. 1–4.

ESTUBLIER, J. Software configuration management: a roadmap. ICSE ’00, may 2000, New York, NY, USA: ACM, may 2000. p. 279–289.

FITZPATRICK, G.; MARSHALL, P.; PHILLIPS, A. CVS Integration with Notification and Chat: Lightweight Software Team Collaboration. CSCW ’06, 2006, New York, NY, USA: ACM, 2006. p. 49–58. . Accessed: 24 jun. 2014.

FUKS, H.; RAPOSO, A.; GEROSA, M. A.; PIMENTEL, M.; LUCENA, C. J. The 3c collaboration model. In: KOCK, N. (Org.). *The Encyclopedia of E-Collaboration*. New York, NY, USA: Information Science Reference, 2007. p. 637–644.

GILBERT, E.; KARAHALIOS, K. LifeSource: Two CVS Visualizations. CHI EA ’06, 2006, New York, NY, USA: ACM, 2006. p. 791–796. . Accessed: 4 may 2014.

GOODMAN, L. A. Snowball Sampling. *The Annals of Mathematical Statistics*, MR: MR124140Zbl: 0099.14203, v. 32, n. 1, p. 148–170, mar. 1961. Accessed: 31 jul. 2014.

GUIMARÃES, M. L.; SILVA, A. R. Improving early detection of software merge conflicts. ICSE 2012, jun. 2012, Piscataway, NJ, USA: IEEE Press, jun. 2012. p. 342–352.

GUTWIN, C.; GREENBERG, S.; ROSEMAN, M. Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation - Springer. In: SASSE, M. A.; CUNNINGHAM, R. J.; WINDER, R. L. (Org.). *People and Computers XI*. London: Springer London, 1996. p. 281–298. . Accessed: 22 jun. 2014.

HOZUMI, T. *Visugit*. Available: <https://github.com/hozumi/visugit>. Accessed: 4 may 2014.

KOIKE, H.; CHU, H.-C. VRCS: integrating version control and module management using interactive three-dimensional graphics. In: 1997 IEEE SYMPOSIUM ON VISUAL LANGUAGES, 1997. PROCEEDINGS, sep. 1997, Isle of Capri, Italy: IEEE, sep. 1997. p. 168–173.

LANZA, M. The Evolution Matrix: Recovering Software Evolution Using Software Visualization Techniques. IWPSE ’01, 2001, New York, NY, USA: ACM, 2001. p. 37–42. . Accessed: 15 feb. 2014.

MURTA, L. G. P. *Gerência de Configuração no Desenvolvimento Baseado em Componentes*. 2006. 213 f. UFRJ, COPPE, Rio de Janeiro, Brasil, 2006.

PRESTON-WERNER, T. *GitHub’s Network Graph*. Available: <https://github.com/blog/39-say-hello-to-the-network-graph-visualizer>. Accessed: 16 feb. 2014.

ROCHKIND, M. J. The source code control system. *IEEE Transactions on Software Engineering. (TSE)*, v. 1, n. 4, p. 364–470, dec. 1975. Accessed: 28 aug. 2012.

SARMA, A.; VAN DER HOEK, A. Palantir: coordinating distributed workspaces. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2002. COMPSAC 2002. PROCEEDINGS. 26TH ANNUAL INTERNATIONAL, aug. 2002, Oxford, United Kingdom: IEEE, aug. 2002. p. 1093 – 1097.

STEINMACHER, I.; CHAVES, A.; GEROSA, M. Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature. *Computer Supported Cooperative Work (CSCW)*, p. 1–46, may 2012. Accessed: 14 oct. 2012.

VOINEA, L.; TELEA, A.; VAN WIJK, J. J. CVSscan: Visualization of Code Evolution. SoftVis ’05, 2005, New York, NY, USA: ACM, 2005. p. 47–56. . Accessed: 23 apr. 2014.

WLOKA, J.; RYDER, B.; TIP, F.; REN, X. Safe-commit analysis to facilitate team software development. ICSE ’09, may 2009, Washington, DC, USA: IEEE Computer Society, may 2009. p. 507–517.

1. http://svnnotifier.tigris.org/ (2012) [↑](#footnote-ref-1)
2. https://github.com/pocorall/scm-notifier (2012) [↑](#footnote-ref-2)
3. http://tools.tortoisesvn.net/CommitMonitor.html (2013) [↑](#footnote-ref-3)
4. http://code.google.com/p/svnradar/ (2011) [↑](#footnote-ref-4)
5. http://www.fsmpi.uni-bayreuth.de/~dun3/hg-commit-monitor (2009) [↑](#footnote-ref-5)
6. http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx (2013) [↑](#footnote-ref-6)
7. http://svnnotifier.tigris.org/ (2012) [↑](#footnote-ref-7)
8. https://github.com/pocorall/scm-notifier (2012) [↑](#footnote-ref-8)
9. http://tools.tortoisesvn.net/CommitMonitor.html (2013) [↑](#footnote-ref-9)
10. http://code.google.com/p/svnradar/ (2011) [↑](#footnote-ref-10)
11. http://www.fsmpi.uni-bayreuth.de/~dun3/hg-commit-monitor (2009) [↑](#footnote-ref-11)
12. http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx (2013) [↑](#footnote-ref-12)