A Holistic, Proactive and Novel Approach for Pre, During and Post Migration Validation from Subversion to Git

Vinay Singh1, Mohammed Alshehri2,*, Alok Aggarwal1, Osama Alfarraj4, Purushottam Sharma5 and K. R. Pardasani6

1School of Computer Science, University of Petroleum & Energy Studies, Dehradun, India
2Department of Information Technology, College of Computer and Information Sciences, Majmaah University, Majmaah, 11952, Saudi Arabia
3School of Computer Science, University of Petroleum & Energy Studies, Dehradun, India
4Computer Science Department, Community College, King Saud University, Riyadh, Saudi Arabia
5Amity School of Engineering & Technology, Amity University, Uttar Pradesh, Noida, India
6Department of Mathematics, Bioinformatics & Computer Applications, Maulana Azad National Institute of Technology, Bhopal, 462003, India

*Corresponding Author: Mohammed Alshehri. Email: ma.alshheri@mu.edu.sa
Received: 31 July 2020; Accepted: 13 September 2020

Abstract: Software development is getting a transition from centralized version control systems (CVCSs) like Subversion to decentralized version control systems (DVCDs) like Git due to lesser efficiency of former in terms of branching, fusion, time, space, merging, offline commits & builds and repository, etc. Git is having a share of 77% of total VCS, followed by Subversion with a share of 13.5%. The majority of software industries are getting a migration from Subversion to Git. Only a few migration tools are available in the software industry. Still, these too lack in many features like lack of identifying the empty directories as pre-migration check, failover capabilities during migration due to network failure or disk space issue, and detailed report generation as post-migration steps. In this work, a holistic, proactive and novel approach has been presented for pre/during/post-migration validation from Subversion to Git. Many scripts have been developed and executed run-time over various projects for overcoming the limitations of existing migration software tools for a Subversion to Git migration. During pre-migration, none of the available migration tools has the capability to fetch empty directories of Subversion, which results in an incomplete migration from Subversion to Git. Many Scripts have been developed and executed for pre-migration validation and migration preparation, which overcomes the problem of incomplete migration. Experimentation was conducted in SRLC Software Research Lab, Chicago, USA. During the migration process, in case of loss of network connection or due to any other reason, if migration stops or breaks, available migration tools do not have capabilities to start over from the same point where it left. Various Scripts have been developed and executed to keep the migration revision history in the cache (elastic cache) to start from the same point where it was left due to connection failure. During post-migration, none of the available version control migration tools generate a detailed report giving information about the
total size of source Subversion repositories, the total volume of data migrated to destination repositories in Git, total number of pools migrated, time taken for migration, number of Subversion users with email notification, etc. Various Scripts have been developed and executed for the above purpose during the post-migration process.

**Keywords:** Databases and information systems; subversion; Git; version control system; trunk; tag; translational settings; author mapping

1 Introduction

For almost all software projects, source code is like the crown jewels, a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful efforts. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences. Software developers working in teams are continually writing new source code and changing existing source code [1]. The code for a project, app, or software component is typically organized in a folder structure or file tree. One developer in the team may be working on a new feature. In contrast, another developer fixes an unrelated bug by changing code; each developer may make their changes in several parts of the file tree. While it is possible to develop software without using any version control, doing so subjects the project to a considerable risk that no professional team would be advised.

There are various version control systems in use like Subversion (SVN), Mercurial, CVS, Git and many more. By far, the most widely used modern version control system is Git. Git is a mature, actively maintained open source project initially developed in 2005 by Linus Torvalds, developer of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open-source. Git has a variety of advanced features like distributed nature, offline commits facility, fast processing, staging feature and many more. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments). In the recent few years, Git has been the first choice as version control of small to massive IT corporate world due to its better features [2].

Git is having a share of 77% of total VCS, followed by Subversion with a share of 13.5% [3]. The majority of software industries are getting a migration from Subversion to Git. Only a few migration tools are available in the software industry. Still, these too lack in many features like lack of identifying the empty directories as pre-migration check, failover capabilities during migration due to network failure or disk space issue and detailed report generation as post-migration steps. In this work, a holistic, proactive and novel approach has been presented for pre/during/post-migration validation from Subversion to Git. Many scripts have been developed and executed run-time over various projects for overcoming the limitations of existing migration software tools for a Subversion to Git migration. Experimentation was conducted in SRLC Software Research Lab, Chicago, USA.

The rest of the paper is organized as follows. Section 2 gives, in brief, the major work done by earlier researchers in the migration process of Subversion to Git VCS. Pre-migration validation and migration preparation are provided in Section 3. Section 4 deals with the process of migration. Section 5 the post-migration validation–report generation & notification of version control migration. Finally, work is concluded in Section 6.
2 Related Work

In Subversion, many earlier studies focus on commit size distribution, which gives the probability that a given commit is of a particular size. Commit size for the number of files committed for a revision can follow power laws [4], Pareto model [5], Petro-distribution [6], etc. Commits have been categorized based on various features, mainly size and comment [7]. Dynamics of open-source software developer's commit behavior have been investigated for Subversion in Chen et al. [8]. It is evidence that within the life cycle and each release of project data sets of project-level and file-level collective commit interval follows a power-law distribution. Four open-source software projects on Apache.org have been considered for the above investigation. An empirical study on inter-commit times in Subversion has been performed in Wang et al. [9] on two projects written in Java. It is observed that both POI and Tomcat distribution of commit intervals follow power laws. Further, two major factors that cause a very long period of inactivity are active committer's behavior of long vacations.

For a centralized VCS, a commit signing mechanism has been proposed in Kumar et al. [10]. The proposed mechanism supports various features like it allows clients to work on a disjoint set of files without retrieving other’s changes. It also allows working with a subset of the repository. Collaborative vocabulary development has been focused on many earlier works [11]. Applicability of Git for collaborative vocabulary development has been investigated in Vaidya et al. [12]. Git4Voc has been proposed, which gives guidelines on how Git can be adopted to vocabulary development. It is shown that by using vocabulary-specific features, Git hooks can be implemented to go beyond the Git plain functionality. LHCb migration from Subversion to Git has been presented in Halilaj et al. [13]. Issues related to specific requirements of LHCb have been addressed. Technical details of migration of large non-standard SVN repositories have also been addressed. It has been claimed that this migration from Subversion to Git has resulted in increased productivity in terms of new projects & the number of contributions and code quality in terms of testing and reviews. A mechanism for classification and extraction of changes is proposed in Diane et al. [14]. A tool named Git Change Classifier (GCC) is proposed, which uses text mining for determining the type of change and regular expressions for extracting the changes. Year-wise number of changes for a file have been reported by GCC where changes have been classified as: bug repairing changes (BRC), feature introducing changes (FIC) and global changes (GC). Various challenges and confusion in learning version control with Git have been reported in Li et al. [15], followed by recommendations for dealing the same. General concepts of centralized and distributed VCS have been discussed in Tanwar et al. [16–20] and how these concepts are implemented by Subversion and Git VCS.

3 Pre-Migration Validation and Migration Preparation

There is no software solution for pre-identification and pre-validation of a complete project structure and placing a version control place holder file created by Git users as a Git repository preserves an otherwise empty project directory. Subversion supports empty repositories, but Git does not as part of the project structure, which might be essentially required for complete software development like lib (libraries), dist. and target folders. If the source code repository has few empty folder structures [21] in Subversion and a Subversion to Git migration is executed, then it would never be completely imported to Git. These empty repositories might be an essential part of the project development structure, for example, lib or dist. folders that are needed to download the maven-based dependencies at compile or run time. In real-time, since these available migration tools never fetch empty repositories [22], it eventually results in incomplete Subversion to Git migration. It would require project build failure; hence whole project delivery pipeline is failed, require manual efforts to correct the project structure in Git repositories and also a start over of the Subversion to Git migration. It results in depletion of manual efforts, industry
timelines and financial losses. These migrations should have pre-migration capability to scan and analyses the entire Subversion repository and to create a Git place holder during Git clone. In this work, many shell Scripts have been developed for identifying all directories/subdirectories which are empty and need a place holder for complete migration.

3.1 Pre-Requisites

Pre-requisites for pre-migration validation and migration preparation are as follows:

- Install Oracle JRE 1.8 or newer (LTR release)
- Install SVN Mirror add-on migration tool in Git BitBucket
- Sufficient space should be available on the BitBucket server as per the size of SVN repositories
- SVN repo URLs must be accessible from the BitBucket server

Dependencies:

Dependencies for pre-migration validation and migration preparation are as follows:

- License procurement for SVN Mirror add-on migration tool from SubGit organization
- Count of SVN users to be migrated to Git

3.2 Migration Workflow

The migration workflow is given in Fig. 1. Steps for Pre-migration validation and migration preparation:

The project structure needs to be validated in Subversion to identify if there is an empty directory in all repositories. For this purpose, a script, shown in Fig. 2 has been developed. Next, Subversion repositories need to be validated for empty project structures using the script shown in Fig. 2.

3.3 Results

Email notification shall be sent by the Script and also results in command line.

Total 3 folders have been fixed by adding place holder (.gitkeep).

3.4 Installation Steps for SVN Mirror Add on

Firstly, a fresh Git project need to be created in Git Bitbucket. Once the project is created then it need to be identified if there is any empty repository. SVN-Mirror add on the tool is to be installed to migrate the project. For this purpose, the following command is executed. subgit.bat configure-layout auto-trunk TRUNK SVN_URL GIT_REPO

Fig. 3 shows the Script for installation and configuration of the SVN Mirror (SubGit) tool.

3.5 Configure SVN Mirror Settings

Now mirror Git repository need to be configured. For this purpose following command is executed to configure Git repository to reflect SVN project: $ subgit configure—layout auto—trunk trunk SVN_PROJECT_URL repos.git

The above command will detect branches layout in the SVN project and then will create an empty Git repository ready to mirror the SVN project.
Figure 1: Flowchart for SVN to Git migration

Figure 2: Script for identifying and filling the empty repository in Subversion project

```bash
#!/bin/sh

# script_name = Pre_migration_validation_script.sh
(TM This function is written to proactively find the empty repositories in SVN and commit those repo with git place holder.

Find_Empty_Svn_Repos()
{
    for repo in `ls -R /var/svn-repos`;
    do
        svn checkout --ignore-externals https://domain/svnrepo/branches/Master repos;
        cd repos;
        find . -type d -empty -not -path "*/svn/**" -exec echo () \;
        find . -type d -empty -not -path "*/svn/**" -exec touch [/].gitkeep \ -exec echo [] \;
        git add --force .
        git commit -m "Add .gitkeep files into empty directories"
        cd ..;
        done;

    for url in `cat urls.txt`;
    do
        svn checkout --ignore-externals url $(basename url);
        cd $(basename url);
        find . -type d -empty -not -path "*/svn/**" -exec touch [/].gitkeep \ -exec echo []/.gitkeep \ ;
        git add --force .
        svn commit -m "Add .gitkeep files into empty directories"
        rmail -a "Empty SVN Repo" superadmin@gmail.com "Hello Admin!! All empty repos have been committed back with git place holder thanks" cd ..
    done;
}
```
3.6 Author Mappings, Adjust Translation & Initial Translation

Here repositories name is to be provided, which subgit tool will use to copy from SVN. This input is very much required because sometime it might not be needed to migrate all repositories from SVN. The same way the authors, which means SVN users, will be filtered out based upon the repositories chosen in repos.git.

Finally, repositories shall be imported by executing an import command. It will start the initial translation and will import all repositories from Subversion to Git.

edit repos.git/subgit/config
$ edit repos.git/subgit/authors.txt
$ subgit install repos.git
$ subgit import repos.git

4 During Migration

In the available migration software tools from Subversion to Git, in case of network failure or for any other reason, if version migration is failed or hampered, then there is no way by which an alert can be sent to the project administrator about network failure. Some of the repositories are very old with giga/terabytes of data. Further, during migration in case of loss of network connection or due to any other reason, if migration stops or breaks, available tools do not have capabilities to start over from the same point where it left. So there must be a mechanism to alert or inform migration administrators in the form of email communication. Version control migration tools should have capabilities to notify the users/administrator about network connection failure between source and destination. Also, the tool should start over from the breaking point precisely from the same revision history where it left after connection breaks. The tool must have the capability to keep the migration revisions history in the cache (elastic cache) to start from the same point where it was left due to connection loss.
Most of the time, Subversion repositories and projects are substantial in nature. In a typical scenario, it could be 1–2 TB. This could lead to migrate data in hours based upon network bandwidth. This leads to the development of Scripts for automation and incorporation of these Scripts in Jenkins to continuously check the status of Git repositories during migration and also to restart the migration if migration has stopped due to network connection failure. This status might include: if the size of the repository is being increased, network ping status between Git and Subversion servers, if commits counts are getting increased, etc. Fig. 4 shows the Script developed to automate these tasks and incorporated with Jenkins to schedule it and restart the migration if migration is failed for any reason.

During the migration process, if Git count is not increasing, then most probably it is due to network failure, etc. In this situation, during migration, the migration process should get a start from the last commit point. To accomplish it, a Script is developed in Python for making a check if Git count is not increasing and to restart the migration process in case of failure. This is shown in Fig. 5. Results and validation of scripts are shown in Fig. 6 for checking network connectivity.

**Figure 4:** Script for checking if Subversion & Git count is increasing

```
#!/bin/bash
set -x
gitcountfile=/home/vinay/git/gitcount.txt
svncountfile=/home/vinay/svn/svncount.txt
old_size=/home/vinay/git/git/oldsize.txt
old_size=/home/vinay/svn/svn/oldsize.txt
gitdirmkdir=/home/vinay/git
svndir.mkdirs=/home/vinay/svn
last_git_count=cat(/home/vinay/git/gitcount.txt)
lst_svn_count=cat(/home/vinay/svn/svncount.txt)
last_svn_count=cat(/home/vinay/svn/svn/oldsize.txt)
old_size=cat(/home/vinay/git/git/oldsize.txt)
old_size=cat(/home/vinay/svn/svn/oldsize.txt)

stop=0
while ((${stop} = 0))
do
  if git ls-remote https://github.com/vinayasing685/copy
    then
echo 'Git Connected';
cd $gitdir;
cd ..
mv -r test/
pwd
  git clone https://github.com/vinayasing685/copy
  cd $gitdir
  gitcommit echo $git rev-list --count --branches master
  if [ "$gitcommits" -gt "$last_git_count" ]
    then
      echo $gitcount > $gitcount.txt
      # git drip
      echo $git drip objects -v --human-readable | grep 'size:"
      git commit
      if [ "$gitsize" -gt "$svnsize old git" ]
        then
          echo $svncount > $svncount.txt
          pwd
          else
            python3 sendmail 'Git Size status' 'Git size not increase'
            stop=1
            exit
          fi
        fi
      else
        echo 'Git Not able to Connect:
        python3 sendmail 'Git Connection status' 'Git Not able to Connect'
        stop=1
        exit
      fi
    fi
  if svn info http://192.168.0.101/copy
    then
echo 'SVN Connected';
cd $svndir;
cd ..
mv -r test/
pwd
  svn checkout http://192.168.0.101/copy
  cd $svndir
  svncommits echo $svn info | grep Revision| awk '{print $2}';
```

**Figure 5:** Python script to check if Git count is not increasing and restart the migration in case of failure through email notification

```
import smtplib
import sys
fromaddr = 'vinayasinghchicago@gmail.com'
toaddrs = ['vinayasinghchicago@gmail.com']

# login
smtpobj = smtplib.SMTP('smtp.gmail.com:502')
smtpobj.starttls()
smtplib.login(username, password)
smtplib.sendmail(fromaddr, toaddrs, msg)
smtplib.quit()

```
In case if commit count is not increasing, then an email notification must be sent to the administrator about it. Fig. 7 shows the Script developed for the above purpose. Fig. 8 shows the email notification received from python scripts about migration failure.

5 Post-Migration Validation–Report Generation & Notification of Version Control Migration

After the migration is completed successfully, none of the available version control migration tool generates a detailed report giving information about the total size of source Subversion repositories, the total size of data migrated to destination repositories in Git, the total number of repositories migrated, time taken for migration, number of Subversion users with email notification, etc. Version control
migration tool should be capable enough to send email notification and generate a detailed report to version control admin/software configuration manager or release engineering team with a detailed report about revision & commit history, user mapping and branch mapping, including tagging feature. The tool should be capable enough to provide this information after successful migration. For all these purposes, a shell Script has been developed for the generation of a detailed report of post-migration and is shown in Fig. 9. Fig. 10 shows the algorithm for printing the detailed report after migration. Fig. 11 shows the Script developed for comparing the Git users with Subversion users while Fig. 12 shows the algorithm for the same. A detailed report generated about migration success is shown in Fig. 13.

```
#!/bin/bash

gitdir=/c/svn_git_repositories/git/patient_clinical_evaluation/
git_developer_dir=/c/svn_git_repositories/git
svn_master_dir=/c/svn_git_repositories/svn/patient_clinical_evaluation/
svn_developer_dir=/c/svn_git_repositories/master
svnrepo=/c/svn_git_repositories/svn/patient_clinical_evaluation/
envir = "Comprehensive Report of on SVN to Git Migration""""

for git in $gitdir
    git log | grep "Author:" | wc -l
for svn in $svn_master_dir
    svn log | grep "r" | wc -l
for envir
    git log | grep "Authors:" | wc --unique

envir = "First and Last commits in Develop Branch"

for git
    git log $(git log --reverse --pretty=format:"%h") | grep "commit"
for svn
    svn log | grep "r" | cut -d " " -f 2 | cut -d " " -f 3
for envir
    git log --pretty=oneline | tail -n 1
for git
    git log $(git log --reverse --pretty=format:"%h") | grep "commit"
for svn
    svn log | grep "r" | cut -d " " -f 2 | cut -d " " -f 3
for envir
    git log $(git log --reverse --pretty=format:"%h") | grep "commit"
for svn
    svn log | grep "r" | cut -d " " -f 3
```
1. Begin
2. Define :gitdir
3. Define :git_developer_dir
4. Define :svn_master_dir
5. Define :svn_developer_dir
6. Define svnrepo: Path
7. Set :gitdir
8. Set :git_developer_dir
9. Set :svn_master_dir
10. Set :svn_developer_dir
11. Set :svnrepo: Path
12. Change Directory :gitdir
13. Print : total Authors in git
14. Change Directory :svn_master_dir
15. Print : total count of Authors in SVN
16. Change Directory : gitdir
17. Print : total count of Authors in Git
18. Change Directory :svn_master_dir
19. Print : The name of all Authors in SVN
20. Change Directory : gitdir
21. Print : The name of all Authors in Git
22. Print : The First commit in Git, SVN
23. Print : The First commit date in Git, SVN
24. Print : The last commit in Git, SVN
25. Print : The last commit date in Git, SVN
26. End

Figure 10: Algorithm for printing the detailed report after migration

```bash
#!/bin/bash
svn_repo_url = "/Users/shashibisht/svn_tutorial/\$
git_repo_url = "/Users/shashibisht/git_tutorial/\$
if [-f SvnAuthors ]
    then rm SvnAuthors
fi
if [-f GitAuthors ]
    then rm GitAuthors
fi
cd $svn_repo_url
svn log --quiet | grep "^r" | awk '{print $3}' | sort | uniq > SvnAuthors
echo "Added list of SVN authors to SvnAuthors file" > logfile
cd $git_repo_url
git log --all --format='@aN' | sort -u > GitAuthors
echo "Added list of GIT authors to GitAuthors file" > logfile
diff -q SvnAuthors GitAuthors >/dev/null
cmp_value=1
if [ $cmp_value -eq 0 ]
then
    echo "SVN and GIT Authors list do not match" > logfile
    mailx -s "SVN Author list not same as GIT Author list" abc@gmail.com
else
    echo "SVN and GIT Authors list match!!" > logfile
    mailx -s "SVN Author list is same as GIT Author list!!" abc@gmail.com
fi
```

Figure 11: Script for comparing the Git users with Subversion users
6 Conclusion

Software development is getting a transition from centralized version control systems like Subversion to decentralized version control systems like Git due to lesser efficiency of former in terms of branching, fusion, time, space, merging, offline commits & builds and repository, etc. Only a few migration tools are available.
in the software industry. Still, these too lack in many features like lack of identifying the empty directories as pre-migration check, failover capabilities during migration due to network failure of disk space issue, and detailed report generation as post-migration steps. In this work, a holistic, proactive and novel approach has been presented for pre/during/post-migration validation from Subversion to Git. Many scripts have been developed and executed run-time over various projects for overcoming the limitations of existing migration software tools for a Subversion to Git migration. During pre-migration, none of the available migration tools have the capabilities to fetch empty directories of Subversion, which results in an incomplete migration from Subversion to Git. In this work, many Scripts have been developed and executed for pre-migration validation and migration preparation, which overcomes the problem of incomplete migration. During the migration process, in case of loss of network connection or due to any other reason, if migration stops or breaks, available migration tools do not have capabilities to start over from the same point where it left. Various Scripts have been developed and executed to keep the migration revision history in the cache (elastic cache) to start from the same point where it was left due to connection failure. During post-migration, none of the available version control migration tool generate a detailed report giving information about the total size of source Subversion repositories, the total volume of data migrated to destination repositories in Git, total number of repositories migrated, time taken for migration, number of Subversion users with email notification, etc. Various Scripts have been developed and executed for the above purpose during post-migration process.

With all above mentioned experimentation done in the SRLC Software Research Lab, Chicago, USA it could be stated that it is very much possible and feasible that any version control migration tool can be made more capable and dynamic by providing a holistic approach for the migration for all repositories from Subversion to Git. With the help of automation and Scripts it can be ensured that Subversion repositories have proper structure compatible to the Git structure before actual migration starts and during migration there is proper notification sent to all stakeholders in the IT business. Automation has been done using python and shell Scripts to complete a post migration validation tasks.

Funding Statement: The author extends their appreciation to the Deanship of Scientific research at Majmaah University for the funding this work under Project No. (RGP-2019-26).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References
[1] Y. Ma, Y. Wu and Y. Xu, “Dynamics of open-source software developer’s commit behavior: An empirical investigation of subversion,” in Proc. SAC, ACM, Korea, pp. 1171–1173, 2014.
[2] O. Arafat and D. Riehle, “The commit size distribution of open source software,” in Proc. HICSS, New York, NY: IEEE Computer Society Press, pp. 1–8, 2009.
[3] C. Kolassa, D. Riehle and M. Salim, “A Model of the commit size distribution of open source,” in Proc. SOFSEM, Heidelberg: Springer–Verlag, pp. 52–66, 2013.
[4] R. Samikannu, R. Ravi, S. Murugan and B. Diarra, “An Efficient image analysis framework for the classification of glioma brain images using CNN approach,” Computers, Materials & Continua, vol. 63, no. 3, pp. 1133–1142, 2020.
[5] P. Sharma and K. Saxena, “Application of fuzzy logic and genetic algorithm in heart disease risk level prediction,” International Journal of System Assurance Engineering and Management, vol. 8, no. 2, pp. 1109–1125, 2017.
[6] C. Kolassa, D. Riehle and M. Salim, “A Model of the commit size distribution of open source,” in Proc. SOFSEM, Czech Republic, pp. 52–66, 2013.
[7] P. Sharma, K. Saxena and R. Sharma, “Diabetes mellitus prediction system evaluation using C4.5 rules and partial tree,” in Proc. 4th ICRITO, India: IEEE, pp. 1–6, 2015.
[8] Y. T. Chen, J. J. Tao, L. Y. Liu, J. Xiong and R. L. Xia, “Research of improving semantic image segmentation based on a feature fusion model,” *Journal of Ambient Intelligence and Humanized Computing*, 2020.

[9] W. Wang, Y. T. Li, T. Zou, X. Wang, J. Y. You et al., “A novel image classification approach via Dense-MobileNet models,” *Mobile Information Systems*, vol. 2020, 2020.

[10] M. Kumar, M. Alshehri, R. AlGhamdi, P. Sharma and V. Deep, “A DE-ANN inspired skin cancer detection approach using fuzzy C-means clustering,” *Mobile Networks and Applications*, vol. 25, no. 4, pp. 1319–1329, 2020.

[11] F. Yu, L. Liu, L. Xiao, K. L. Li and S. Cai, “A robust and fixed-time zeroing neural dynamics for computing time-variant nonlinear equation using a novel nonlinear activation function,” *Neurocomputing*, vol. 350, pp. 108–116, 2019.

[12] S. Vaidya, S. T. Arias, R. Curtmola and J. Cappos, Commit signatures for centralized version control systems. in *Advances in Information and Communication Technology*, vol. 562. Cham: Springer, 320–328, 2019.

[13] L. Halilaj, I. Grangel, G. Coskun, S. Lohmann and S. Auer, “Git4Voc: Collaborative vocabulary development based on Git,” *International Journal of Semantic Computing*, vol. 10, no. 2, pp. 167–191, 2016.

[14] J. P. Diane, I. Hillmann and G. Dunsire, “Versioning vocabularies in a linked data world,” *International Journal of Semantic Computing*, vol. 10, no. 2, pp. 167–191, 2016.

[15] T. M. Li, H. C. Chao and J. M. Zhang, “Emotion classification based on brain wave: A survey,” *Human-Centric Computing and Information Sciences*, vol. 10, pp. 238–249, 2019.

[16] R. Tanwar and P. Sharma, “Caught hacker: Curing DDOS attack,” in *Proc. ICTCS*, New York, NY, USA: ACM, pp. 4–11, 2016.

[17] I. Zaikin and A. Tuzovsky, “Owl2vcs: Tools for distributed ontology development,” in *Proc.*, OWLED: Citeseer, 2013.

[18] M. Clemencic, B. Couturier, J. Closier and M. Cattaneo, “LhcB migration from subversion to Git,” *Journal of Physics*, vol. 898, pp. 1–4, 2017.

[19] A. Kaur and D. Chopra, “GCC-Git change classifier for extraction and classification of changes in software systems,” in *Proc. ICCT-LNNS*, Singapore: Springer, 19, pp. 259–267, 2018.

[20] V. Isomottonen and M. Cochez, “Challenges and confusions in learning version control with Git,” *Communications in Computer and Information Science*, vol. 469, pp. 178–193, 2014.

[21] S. Mishra, S. K. Sharma and M. A. Alowaidi, “Analysis of security issues of cloud-based web applications,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 3, no. 1, pp. 50, 2020.

[22] S. K. Sharma, “An empirical model (EM: CCO) for clustering, convergence and center optimization in distributive databases,” *Journal of Ambient Intelligence and Humanized Computing*, 2020.