Secure Extended CVS for Academia

ShaileshHule, Gopi Krishna Kapagunta, Jerry Antony, Shreyas Kulkarni, Ashwin Bhide

***Abstract*— During the academic lifespan, almost all students find it difficult to manage and backup the coding assignments conducted during their practical. Although students are instructed to create a folder on the computer and save all their assignments into it; there is no guarantee that the folder will not get deleted from the computer until the students next turn arrives. Also sometimes only partial program is completed in the college and remaining program needs to be done at home which is again a cause of concern. Also teachers cannot access all student coding assignments at one place and hence find it difficult to evaluate the student's performance. Our system is developed to cater to all this needs. Our system will include an Eclipse plug-in as well as a web site with users having both the options to backup their assignments. Although Github, a product on similar lines already exists, it does not provide specific functionality needed for academia. Our system is specifically developed for colleges to simplify task of students as well as teachers. It can be used as self-contained application for code assignment backup and management.**

***Index Terms*— Academia, CVS, Plug-in, Repository.**

I. INTRODUCTION

Every science related academic institution has courses which need students to perform lab assignments in the form of source codes. The problem statements is same for a particular assignment, but with different implementations for each student. The physical resources, viz. the labs, are reused over various batches of courses. But storing of source codes in the laboratory computers exposes it to security threats, and makes the evaluation of the assignments difficult. Also, it restraints the availability of the source code with the author. Thus there is a need of a system which provides security to the source codes, and providing a central repository so as to increase the availability of the source code to the author. During the academic lifespan, almost all students find it difficult to manage and backup the coding assignments conducted during their practical. Although students are instructed to create a folder on the computer and save all their assignments into it; there is no guarantee that the folder will not get deleted from the computer until the students next turn arrives. Also sometimes only partial program is completed in the college and remaining program needs to be done at home which is again a cause of concern. Also teachers cannot access all student coding assignments at one place and hence find it difficult to evaluate the student's performance. Our system is developed to cater to all this needs. Our system will include an Eclipse plug-in as well as a web site with users having both the options to backup their assignments. Although Github, a product on similar lines already exists, it does not provide specific functionality needed for academia. Our system is specifically developed for colleges to simplify task of students as well as teachers. It can be used as self-contained application for code assignment backup and management**.**

Manuscript received March 04, 2015.

Prof. Shailesh Hule, Computer Department, Pune University/ Pimpri Chinchwad College of Engineering, Pune, India, +919226094987,
Gopi Krishna Kapagunta, Computer Department, Pune University/ Pimpri Chinchwad College of Engineering, Pune, India, +919028418861,
Jerry Antony, Computer Department, Pune University/ Pimpri Chinchwad College of Engineering, Pune, India, +919405253597,
Shreyas Kulkarni, Computer Department, Pune University/ Pimpri Chinchwad College of Engineering, Pune, India, +919764674232
Ashwin Bhide, Computer Department, Pune University/ Pimpri Chinchwad College of Engineering, Pune, India, +919766940333,

A. Version Control

Version Control Systems (VCS), Revision Control Systems (RCS), Software Configuration Management, Source Code Management (SCM) or Source Code Control may sound different, but are basically the same thing, i.e. they are systems that enable the acceleration and simplification of the software development process , and enable new workflows. They keep track of files and their history and have a model for concurrent access. For the project, we have referred several published papers, on the topics of RCS and CVS. We have read about the different types of methods used to store versions and different types of concurrency control mechanisms.
There are 2 approaches to implementing VCS:

- The centralized model, with the Centralized Version Control Systems (CVCSs)
- The distributed model, with the Distributed Version Control Systems (DVCSs)

Every stage in the history of a file is identified by a revision or version(typically an integer). A revision consists of the file and some metadata. The metadata that is stored can differ depending on the program. The newest revision is often called head. The user can check out single file as well as the entire repository and he can specify the revision he wants to retrieve from the repository. By doing a check out the user retrieves a working copy on his local computer. A working copy does not include any history but it enables the user to edit the files. After editing the files the user commits the changed files to the repository. He also adds a commit log or commit message, which is a description of what has changed since the last version. By committing the changed file from the working copy into the repository a new revision is created. The revision number rises.

B. DVCSs and CVCSs

The fundamental difference between DVCSs and CVCSs is that DVCSs do not require a central server that contains the repository. Every user has the complete repository on their local computer, it is called local repository. There is no need for a heavy weight server program. In DCVSs, commits, branches, tags, and checkouts fulfill the same tasks as in centralized environments except that they only communicate with the local repository instead of with the central repository.

The classic approach in scenarios that contain concurrency is the lock-modify-unlock mechanism. It is often called pessimistic approach. A resource gets explicitly locked by a process. Only this process can modify their source. After modifying the resource the lock gets removed. It guarantees, that no merging is needed, because a file can be checked out by only one user. This approach does not work for VCSs. The users tend to forget to remove the lock and the file cannot be accessed by anybody.

C. VCSs

VCSs work with the copy-modify-merge mechanism. It is often called optimistic approach. A resource can be copied and modified by more than one process but may be merged afterward. The optimistic approach proved to be better suited for VCSs. It required less administration and allows a better workflow. Although, some CVCSs offer also the lock-modify-unlock mechanism, because it can be helpful in some scenarios. VCSs need a space efficient way to store different versions of the same file. Delta encoding is a way to store only the changes of different versions of a file instead of the entire file. The difference is called diffordelta. Delta encoding is very efficient for source code because the part of the source code that changes is normally relatively small.

In all the current systems that we have seen, the implementation of the VCS has been through files. In our approach, we are going to use MongoDB to store the codes, as MongoDB is a document-based database tool. For the present scope, we are going to implement a Revision Control System without concurrency control, but keeping in mind the scalability, the future scope would be of a Concurrent Versioning System.

II. BACKGROUND

A. RCS for versioning control

Version control is the task of keeping software systems consisting of many versions and configurations well organized. The Revision Control System (RCS) is a set of UNIX commands that assist with that task. RCS also offers facilities for merging updates with customer modifications, for distributed software development, and for automatic identification. Identification is the stamping of revisions and configurations with unique markers. These markers are akin to serial numbers, telling software maintainers unambiguously which configuration is before them.

Suppose a text file f.c is to be placed under control of RCS. Invoking the check-in command

cif.c creates a new revision group with the contents of f.c as the initial revision (numbered 1.1) and stores the group into the file f.c,v. Unless told otherwise, the command deletes f.c. It also asks for a description of the group. The description should state the common purpose of all revisions in the group, and becomes part of the groups documentation. All later check-in commands will ask for a log entry, which should summarize the changes made.

Files ending in ,v are called RCS files (v stands for versions); the others are called working files. To get back the working file f.c in the previous example, execute the check-out command: co f.c

This command extracts the latest revision from the revision group f.c,v and writes it into f.c. The file f.c can now be edited and, when finished, checked back in with ci: ci f.c.

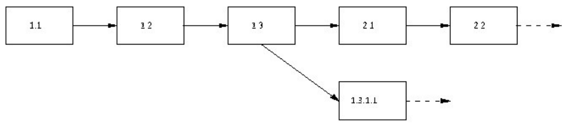

Fig. 1. A revision tree with one side branch

B. Distributed development and customer modifications

Assume a situation as in Figure , where revision 1.3 is in operation at several customer sites, while release 2 is in development. Customer sites should use RCS to store the distributed software. However, customer modifications should not be placed on the same branch as the distributed source; instead, they should be placed on a side branch. When the next software distribution arrives, it should be appended to the trunk of the customers RCS file, and the customer can then merge the local modifications back into the new release. In the above example, a customer's RCS file would contain the following tree, assuming that the customer has received revision 1.3, added his local modifications as revision
1.3.1.1, then received revision 2.4, and merged 2.4 and 1.3.1.1, resulting in 2.4.1.1

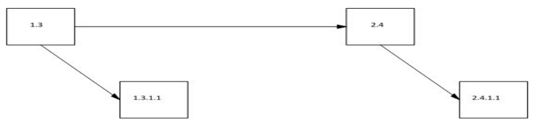

Fig. 2. A customers revision tree with local modifications

C. Representation of revisions as deltas

A delta is a sequence of edit commands that transforms one string into another. The deltas employed by RCS are line-based, which means that the only edit commands allowed are insertion and deletion of lines. If a single character in a line is changed, the edit scripts consider the entire line changed.

Using deltas is a classical space-time trade-off: deltas reduce the space consumed, but increase access time. However, a version control tool should impose as little delay as possible on programmers. Excessive delays discourage the use of version controls, or induce programmers to take shortcuts that compromise system integrity. The most recent revision on the trunk is stored intact. All other revisions on the trunk are stored as reverse deltas. A rev erse delta describes how to go backward in the development history: it produces the desired revision if applied to the successor of that revision. This implementation has the advantage that extraction of the latest revision is a simple and fast copy operation.

There are several techniques for delta application. The naive one is to pass each delta to a general-purpose text editor. A prototype of RCS invoked the UNIX editor ed both for applying deltas and for expanding the identification markers.

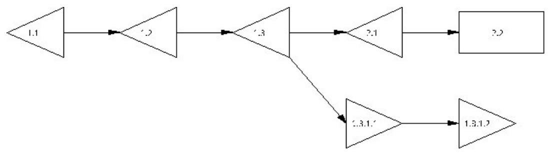

Fig. 3. A revision tree with reverse and forward deltas.

III. SYSTEM ARCHITECTURE

During the academic lifespan, almost all students find it difficult to manage and backup the coding assignments conducted during their practical. Although students are instructed to create a folder on the computer and save all their assignments into it; there is no guarantee that the folder will not get deleted from the computer until the students next turn arrives. Also sometimes only partial program is completed in the college and remaining program needs to be done at home which is again a cause of concern. Also teachers cannot access all student coding assignments at one place and hence find it difficult to evaluate the student’s performance. Our system is developed to cater to all this needs. SECA consists of various modules which increases security of the files and also stores the files in an optimal structured manner to reduce the load on database. Wtih our system students can easily rollback to any older version and need not maintain different folders for the same.

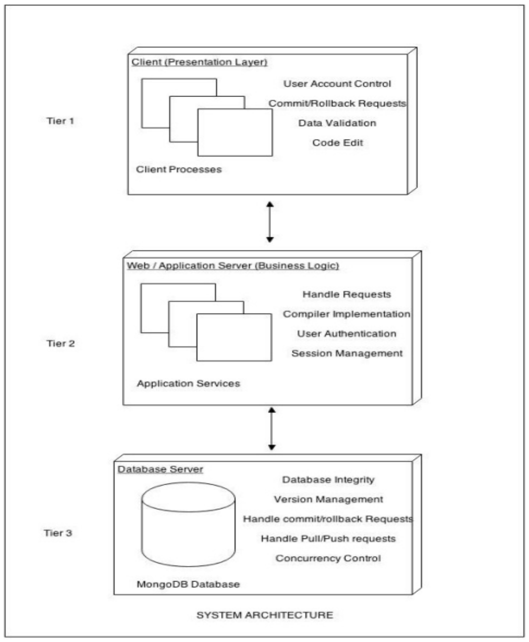

Fig. 4. SECA System Architechture Diagram

Our system will have following modules:-

- Eclipse Plugin
- Web Interface
- Web Server

Database Server

A. Eclipse Plug-in

The eclipse plug-in will consist of a CVS perspective, and additional views for the same. The views will include a project explorer to explore the current workspace, a repository explorer to view/push/pull all the repositories belonging to the user embedded into the environment of the Eclipse IDE. There will also be a view to login to the system, and to display the information related to the current user. There will also be a view to view the different versions of a file and multiple code editing windows, along with a console, a standard output window and an error-log window to display errors encountered during the use of the system.

B. Web Interface

The web interface will consist of a homepage with an option to login to the system either as a student or a tutor, with their respective dashboards. Students will have read/write access to all their codes via the website as an online editor will be provided, complete with a provision to compile the code on the web server and test it. The student will also be able to download the code, and view the previous revisions of code. The tutor will have read/execute access to the codes of all the students who are taught by the tutor. He can specify the test cases for a particular assignment and check if the code of a particular student executes properly for the given test case.

Web Server

The server-side of the client-server model would be interfaced with at least two different servers at the backend, one being the web server which would handle all requests by the client systems both from the eclipse plugin and the web-interface, and the other server being a file server which would work for the exclusive purpose of handling the repositories of the users and maintaining the revisions of all the files, providing safeguard mechanisms against crashes. To keep the system lag free more than one file server would be desired so as to make sure all requests are handled in real-time

C. Database Server

The database server is responsible for interfacing with the web server to execute any queries from the client modules, and also responsible for managing the revision control system, calculating and storing the deltas and storage of files. It is used to store the repositories and providing the data whenever required, and implementing safeguard mechanisms like backups, redundancy to cope up with server crashes.

IV. Conclusion

Hence the system provides the essential security, portability and systematic evaluation portal needed for coding assignments in Academia**.**

References

[1] Stefan Otte, ”Version Control Systems” , 2010.

[2] Walter F.Tichy, ”RCSA System for Version Control ”, 1987.

[3] Gerardo Canfora, Luigi Cerulo and Massimiliano Di Penta, ”Identifying Changed Source Code Lines from Version Repositories”, in MSR’07: Fourth International Workshop on Mining Software Repositories, IEEE Computing Society.

[4] Parminder Kaur and Hardeep Singh, ”A Model for Versioning Control Mechanism in ComponentBased Systems”, In ACM SIGSOFT Software Engineering Notes September 2011 Volume 36 Number 5.

Prof. Shailesh Hule has Bachelor’s Degree in CSE and Master’s Degree in IT. Has keen interest in fields like Software Engineering, Operating Systems and Algorithms. He also has more than six years of teaching experience.

Gopi Krishna Kapagunta, Bachelor of Engineering, Student, Primpri Chinchwad College of Engineering, Nigdi, Pune.
Area of Interest: Embedded Systems.

Jerry Antony, Bachelor of Engineering, Student, Primpri Chinchwad College of Engineering, Nigdi, Pune.
Area of Interest: Applied Computing.

Shreyas Kulkarni,
Bachelor of Engineering, Student, Primpri Chinchwad College of Engineering, Nigdi, Pune.
Area of Interest: Applied Computing.

Ashwin Bhide,
Bachelor of Engineering, Student, Primpri Chinchwad College of Engineering, Nigdi, Pune.
Area of Interest: Applied Computing, Big Data.