

USE OF VERSION CONTROL SOFTWARE TO MANAGE A COMPUTER ARCHITECTURE DESIGN PROJECT BY STUDENT TEAMS

Archana Chidanandan
Computer Science and Software Engineering
Rose-Hulman Institute of Technology
Terre Haute, IN 47803

Laurence D. Merkle
Computer Science and Engineering
Wright State University
Dayton, OH 45435

Abstract

In this paper, we describe the role of version control software in managing projects involving teams of three to four students in the Computer Architecture course. In the course, students learn design principles of computer architecture. They demonstrate and deepen their understanding by designing and implementing their own “miniscule instruction set” processor. They use Electronic Design Automation (EDA) tools such as Cadence, NCSim, and Xilinx ISE Foundation to implement their design, and they are strongly encouraged to use Verilog HDL to implement their hardware components and datapath. In this offering of the course, we required students to use Subversion (SVN) to manage their design files and surveyed them to see how the use of SVN helped them in managing their projects. From their responses, we conclude that the use of SVN greatly facilitated sharing of design files and version control. We believe that the use of this version control software can be extended to other hardware design courses.

Introduction

The use of version control software has often been advocated to manage software engineering projects. It has also been adopted by a number of Computer Science and Software Engineering instructors to assist their students to manage team project materials as explained in [1,2]. These projects typically involve two or more people working together or software that is distributed across many files. Version control software serves a number of purposes in managing such work. For example, it simplifies

how code is shared between members of the team and it provides for a mechanism to organize and save multiple versions of the software.

In our Computer Architecture I course, students work in teams to design and implement a processor. In one project, students are provided the design and code of a modified version of the IJVM (Integer Java Virtual Machine) architecture [3]. They are then required to complete the design by adding the components to support interrupt-based I/O. In another project (henceforth called the term project), students are required to design their own “miniscule” instruction set processor [4] and then model it using EDA software. They are also encouraged to implement it on a Field Programmable Gate Array (FPGA) board.

In both projects, students are required to work in teams of between three and five students. This requires them to share design files and implementation files with each other. As the projects are quite extensive and the work is done over multiple weeks, the students also have to learn to manage all their files while working with a team.

In the Fall 2007 offering of this course, we decided to have students use version control software to manage their projects. We hoped that this would solve some of the problems students had encountered in the past, regarding sharing of code and obtaining older (and possibly correct) versions of their design files.

In this paper, we begin by explaining our motivation in encouraging students to use

version control software. Following that, we discuss how the software was used in the course by our students. In the next section, we describe how we administered our surveys and then present the responses to our survey questions. We next describe our conclusions obtained from the responses and finally describe how the use of the version control software can be extended to other similar projects.

Why use version control software?

In the Computer Architecture I class, students work on projects in teams. The projects span a number of weeks and in some cases almost ten weeks. Students form teams consisting of three to five members. The projects require students to design an Instruction Set Architecture, the register transfer language, datapath and control units, and all necessary components. They finally implement the design using Cadence and Xilinx EDA software tools. Students use the schematic capture tool and specify modules in Verilog HDL. Each milestone has a number of deliverables – sometimes design documents, and in other cases implementation files. The final implementation phase can extend anywhere from two to four weeks. The instructors may also provide students with some parts of the design for either optional or mandatory incorporation into the students' design.

In the past offerings of the course, we have observed the following challenges:

- Ensuring code developed by different team members is interfaced correctly
- Avoiding duplicated effort in code development
- Efficient sharing of code with teammates
- Informing teammates about newer versions of code that they need to integrate with their own code
- Obtaining help from the instructor on code-related issues
- Integrating changes to instructor-provided code that students had already modified and integrated into their project.

The instructors conjecture that some of these issues would be alleviated by using version control software to manage the various documents and the software components that the students worked on over the course of a project. Version control software such as CVS [5] and SVN [6] are the most popular programs currently used. In our department, we are currently using SVN to manage our courses and projects. Henceforth, in this paper, when we refer to version control software we are referring to SVN. However, most of our observations are probably valid for other version control software such as CVS.

Version Control Software in the Computer Architecture project

Here we list the various activities that are performed to manage the project:

- a. An SVN repository is created for each project team by the instructor.
- b. If the instructor intends to provide any code to the students, the repositories are populated with the code.
- c. Students are encouraged to store all the Verilog files (design and test) and other Xilinx source files in the repository.
- d. Each member of the team could work on his or her part of the project and periodically submit the new files or modifications to existing files back to the repository. Team members must also periodically update their repositories to obtain the new files and modifications submitted by their team members.
- e. If a team requires an instructor's assistance with a code related problem, the team must first commit the latest version (or the problematic version) of their files to the repository. Instead of having to receive all the required design files by e-mail or on a thumb drive, the instructor can check out the appropriate version of the files from the team's repository to his/her own computer and examine the code and offer suggestions

to students. The team and instructor need not even be in the same room for the team to obtain help from the instructor. For example, they could communicate via an instant messaging service.

- f. Similarly, if the instructor wants to provide updated versions of files when the team has already integrated files into their design, the instructor can commit the files to the student folders and the SVN tool can help the team identify the difference between two versions of the file and thus select the portions that need to be retained and others that may be discarded.
- g. Students also commit a final version of their files to the SVN repository where it becomes available for instructors or assistants to grade.

Figures 6 and 7 provide examples of how the TortoiseSVN graphical user-interface can be used to perform some of the operations described in this section. Also, described in Figure 8 is how to set up an SVN server on a local PC, in case a centralized server is not available. This procedure may be used to create an SVN server on a Windows machine. An SVN server can be installed similarly on a Linux machine.

Assessing the use of the version control software for the project

Pre- and post-surveys were conducted to determine the effects of the version control software in assisting students while working with relatively large projects. In the pre-survey, students were asked questions related to how they would manage their project if they did not use version control software. They were also asked a series of questions to determine how they expected the use of version control software to assist them in coordinating their team work and also manage the various versions of code they would be writing to implement their design. In the post-survey, administered at the end of the term after students had been using the version control software for a significant period of time, the students were asked to

answer questions about their experiences with the software. The questions and a summary of the answers are provided in the following section.

We surveyed students from three sections of the Computer Architecture I course in the fall term. We had a total of 52 students respond to the pre-survey and 50 students respond to the post-survey. The students were sophomore and junior computer science, computer engineering, and software engineering majors. The course is required for these majors. A substantial majority of the students had experience using the version control software in at least one previous course.

Responses to the two surveys

Students were asked to respond to the first set of questions in the pre-survey assuming that they would not have access to version control software. This set of questions and representative responses follow.

1. Please describe how you would ensure that code developed by different team members interfaced correctly.

- The use of a common location to store the files including ftp sites, or using e-mail to periodically email modified files to all members of the team.
- Have meetings frequently to discuss each person's responsibility and contribution, allowing for individuals to work on their own.
- Wait for all team members to complete before system testing.
- Have only one member work at a time.
- Designing to the last detail to ensure that all team members are aware of the naming conventions and the inputs and outputs of all modules are clearly defined.
- Use some messaging software to keep in contact with any team members that are concurrently working.
- Do all the work on one computer with one person managing all the files and all team

members meeting frequently to help the person complete the work on his/her computer.

2. Please describe how you would avoid duplicated effort in code development.

- Break down the design into smaller modules, create a list of modules to be created and each team member would have to know exactly what their responsibility is in terms of modules to develop.
- Document all the code.
- Constant communication amongst team members to update each other on their progress (using e-mail or an instant messaging system).
- Impossible to prevent duplication of work unless the team members have prior experience working with each other.
- Create a schedule and specify when each module is due and require team member to keep to the schedule.
- Have only one member of the team manage all the code.

3. Please describe how you would share code with your teammates.

- Use a portable hard-drive to copy code to and share with team mates.
- E-mail the code to each team member when done.
- Develop the code [on] only one computer.
- Meet daily to coordinate and share code.
- Use some kind of version control software.

4. Please describe how you would inform your teammates about newer versions of code that you would like them to integrate with their code.

- Email the code with descriptions on what changes have been made and the significance of the module to the project.
- Work together to ensure that every team member is aware of everything occurring in

project.

- Daily team meetings to update files.

5. Please describe how you seek help from your instructor on code-related issues.

- Email problematic code or files to instructor with description.
- Use an instant messaging system to contact the instructor.
- Visit his/her office.
- Use in-class project time and show the code on student's computer (after ensuring that the student code is up to date).

6. Please describe how you would integrate changes to instructor-provided code that you had already modified and integrated into your project.

- Look for the affected code in files and make changes manually.
- Ensure that original files provided by instructor are not modified and all additional code is written in separate files. The new code can then be integrated by simply replacing the original files.

Students were asked to respond to a corresponding set of questions in the post-survey in light of their access to version control software during the project. This set of questions and representative responses follow.

1. Please describe how you feel the use of version control software impacted each of the following.

a. Your ability to share code with your teammates efficiently and reliably.

- SVN was extremely helpful, because we could all look at the same block of code at the same time and test different things about it simultaneously instead of all 4 of us huddling around one computer and wasting a lot of time.

- It worked most of the time. Sometimes there were conflicts and you had to delete everything to make it work because it said there were too many differences or conflicts.
- Did not have to e-mail code or place on a server.
- The comments section when committing a file to the repository, allowed team-mates to not just share files but also specify the updates being made and highlighted the areas that needed more work.
- The fact that there were 150 revisions for one of the teams implies that the team did not have to send 150 e-mails.
- Being able to retrieve older versions of files was helpful.

b. Your instructor's ability to answer your code-related questions efficiently and effectively.

- When a question arose, [the] team could add the problem code into their repo and the instructor could check the code out on his/her computer and respond pretty quickly. Students did not have to go to the Professor's office and could get help even when off-campus.

c. Your ability to integrate updates to instructor-provided code.

- Once the instructor added the updates to the team repositories, students only had to update their local folder.

Figures 1 to 5 show the responses for questions on both the pre- and post-surveys.

Application in Engineering Disciplines

In this section, we describe how the version control software could be used in other engineering disciplines. There are at least two scenarios in which version control software would be useful.

The first is in any programming course that is taught (e.g. Matlab or Maple). Each student or

student team can be provided with a repository on the central server. Any templates and sample code that the instructor wants to provide the students can be placed in each team's repository and students can checkout, modify and update the program files and related documents in their repositories. Instructors can checkout the files at any time from each repository to grade them or to provide feedback and assistance.

The second scenario is for a design course especially one which requires students to work in teams and maybe even across multiple terms. Each team can be provided a repository and students can upload their design documents to the repository and thus share it with each other. This will prevent one person having to manage all the documents as would otherwise be the case and any version of the document can be retrieved at any time to see what changes have been made. As stated earlier, all types of files can be stored in the repository – Office files, CAD design files, programming source files (Matlab, C), text files, and image files.

In order to facilitate this, a faculty member will have to perform the following actions:

- a. Set up a server on a centralized machine that will be up and running all the time (see steps 1-4 of Figure 8 for instructions on setting up a server, and Section 6.2 on space and bandwidth requirements). If a centralized server is not available, individual teams can also set up the server on their own machines. However, the authors would not encourage this procedure for persons using the software for the first time.
- b. Create a repository for each team [11] (e.g. *C:/svn/team01*, *C:/svn/team02*, and so on). See Figure 8, step 3.
- c. Create a username and password for each student as shown in Figure 8, Step 6. Note: The “-c” option when creating user accounts is to create a new password file, so use it only when you are creating the first user for

a repository. Also, specify the appropriate password file for each team by adding the appropriate location in the Apache configuration file, as follows:

```
<Location /svn/team01>
#Specify path
DAV svn
SVNPath c:\svn\team01
#Specify authentication
AuthType Basic
AuthName "Subversion Repository"
AuthUserFile c:\passwd\passwords-1
Require valid-user
</Location>
```

- d. Let each student know their username and password and also the location of their repository (e.g. <http://IP-address-of-server/svn/team01>).
- e. Instruct students to install the SVN client. They are then ready to use the repository.
- f. The instructor should also install the SVN client on his/her local machine to access each repository. In order to have access to each repository, the instructor will have to add his/her username to each of the password files as described earlier.

Conclusions

In this section, we discuss the responses of the students and what can be inferred from them. We also discuss what the instructor needs to do to support and manage such a system.

Student responses

From the responses of the students, we can conclude that the most beneficial aspect of using the version control software is being able to share code easily amongst members of the team. 94% of the students agreed on this point. It is interesting to note that 98% of students anticipated this to be the case before they actually used the software.

Looking at the descriptive answers, it was observed that students did not always divide their work up properly beforehand. This meant that there were times when more than one team member was working on the same file simultaneously. When it came time to commit the files back to the repository, the version control software would detect the conflicts and the commit would fail. This is the desired behavior, but students found it difficult to resolve these conflicts and this led to frustration with the software. Perhaps if the instructor were able to demonstrate to the students using an example of how to resolve conflicts then students may not have similar problems in the following terms.

Students were typically positive about being able to obtain the latest versions of their teammates' code from the repository. A couple of students noted how many revisions they had committed to the repository, and this was an indication of how many e-mails they would have had to send each other with the modified files in the absence of a version control system.

In the pre-survey, 81% of the students believed that the version control software would make it easier to get assistance from the instructor. After working on the project for some time and having had the need to obtain help from the instructors, 94% of the students agreed that the version control system facilitated getting help from the instructor.

Students also agreed that there was less duplication of code as the version control software allowed them to detect it early by reporting conflicts during attempted commit operations. This possibly forced the students to be more disciplined in their approach to division of the work amongst the team members.

As one student wrote, if the version control software was used correctly, then the repository was very useful and made teamwork much more efficient and smooth. Specifically, this meant that each team member was conscientious about

committing their latest code to the repository, writing meaningful and descriptive comments during the commit, and updating their working copy before committing new changes. In addition, one could add that if the team planned their implementation well and had clear naming conventions for files and a clear division of tasks then the version control software was even more effective. In the absence of such planning, the version control software detected the problems early. While this may be a frustrating experience, it is less frustrating than detecting such problems late in the process, not to mention being a valuable learning experience in itself!

Impact on the instructor

From an instructor's perspective, the concerns are

- a. Helping students learn to use the tool
- b. Creating repositories for the teams and populating them as required
- c. Checking out entire projects of student teams and navigating through their files
- d. Maintaining a server to host the repositories

All of these are relatively straightforward tasks. There are plenty of tutorials available for the most popular version control software tools. Most also have GUIs [7-9] that make the process fairly intuitive. The learning curve is not very steep and at Rose-Hulman the introductory computer science courses require students to use version control. Both "b" and "c" are also relatively simple. For example, at Rose-Hulman, our colleagues and system administrators have written a number of scripts that allow us with relative ease to create repositories for a list of students IDs stored in a file. Similarly, there are scripts to populate repositories and checkout repositories. In [10], the authors describe how instructors can use the version control system to manage their own course material and how it can enhance teaching.

In terms of the server requirements, each term our department colleagues and we create an average of 300 new student repositories for the various courses being taught. For such a load, the SVN server requirements include about 512MB of RAM, and a 100Mb bandwidth network. Even with an average of 300 repositories, the I/O traffic is minimal and a SATA is sufficient for disk access. A really busy server would need several drives in a striped RAID configuration to handle the disk I/O. The disk subsystem would be the first bottleneck, before RAM or CPU.

While we have used the version control software for projects in the Computer Architecture I course, it could be used easily by teams of students working on any type of design project – be it in civil engineering, mechanical engineering or bio-medical engineering. The version control software allows for all kinds of files from programming files to text files to design documents to be stored and retrieved as needed, although it is more effective for plain text files than for binary format files such as Microsoft Office documents. It can support large teams and large projects. In the words of one of the students, "I couldn't think of a better way to share code. Every time you log-in, you update to the current version, commit when [you're] done and everyone is on the same page!"

References

1. K. Reid and G.V. Wilson, "Learning by Doing: Introducing Version Control as a Way to Manage Student Assignments," in Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, pages 272–260. ACM, ACM Press, 2005.
2. K. T. Hartness, "Eclipse and CVS for group projects," in Proceedings of CCSC'06, Consortium for Computing Sciences in Colleges, 2006.

3. A.S. Tanenbaum, *Structured Computer Organization*, 5th ed., Prentice Hall, 2006.
4. Archana Chidanandan, J.P. Mellor, and Laurence D. Merkle, "Design and Implementation of a Minuscule General Purpose Processor in an Undergraduate Computer Architecture Course," in Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education (MSE 2007).
5. P. Cederqvist, "CVS—concurrent versions system," <http://ximbiot.com/cvs/manual/cvs-1.11.22/cvs.html>, 2006, (accessed September 2008).
6. B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, O'Reilly, 2004.
7. <http://tortoisesvn.tigris.org/> (accessed September 2008).
8. <http://scplugin.tigris.org/> (accessed September 2008).
9. <http://rapidsvn.tigris.org/> (accessed September 2008).
10. Curtis Clifton, Lisa C. Kaczmarczyk, and Michael Mrozek, "Subverting the fundamentals sequence: Using version control to enhance course management," in Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, pages 86–90. ACM, ACM Press, 2007.
11. <http://www.subversionary.org/howto/setting-up-a-server-on-windows> (accessed January 2009).
12. <http://www.apache.org/dist/httpd/binaries/win32/> (accessed January 2009).
13. <http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91> (accessed January 2009).

Biographical Information

Archana Chidanandan received a B.E. in Electronics and Communications Engineering in 1997 from the College of Engineering, Anna University, India. She also received the M.S. and Ph. D. in Computer Engineering from the University of Louisiana at Lafayette in 1999 and 2004 respectively. She has worked as a systems engineer at Tata Consultancy Services, India and is currently an Associate Professor of Computer Science and Software Engineering at Rose-Hulman Institute of Technology, where she teaches courses such as Computer Architecture, Operating Systems, Computer Networks, Computer Security, and Mobile Computing.

Larry Merkle received a B.S. in 1987 from Rensselaer Polytechnic Institute, as well as a M.S.C.E in 1992 and a Ph.D. in 1996 from the Air Force Institute of Technology. In the Air Force, he worked in AI, parallel algorithms, evolutionary computation, computational science and engineering, and basic research. He has taught at the United States Air Force Academy and Rose-Hulman Institute of Technology, and held several adjunct positions. He has advised 8 senior thesis students, has taught 22 different courses, and has over 65 publications and presentations. He is currently the assistant chair of Wright State University's Department of Computer Science and Engineering.

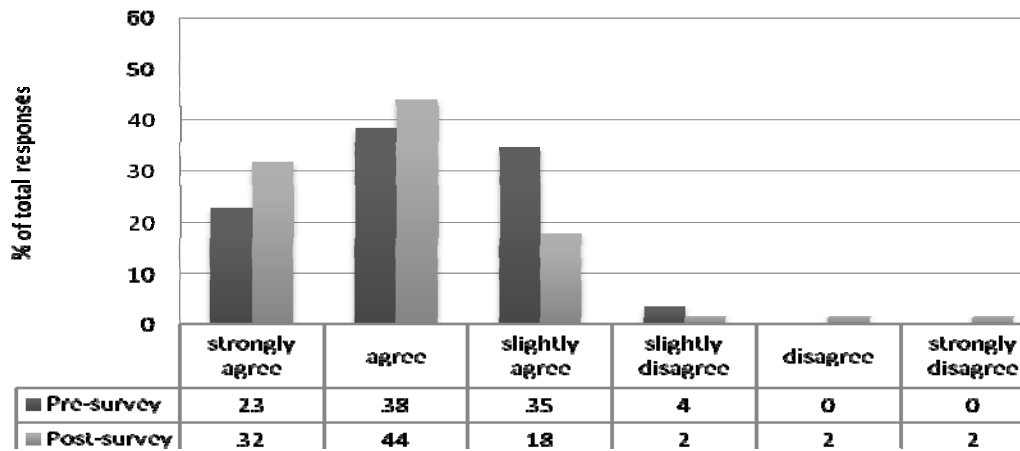


Figure 1. Response to "Version control software will help ensure that my components will interface correctly with those of my teammates."

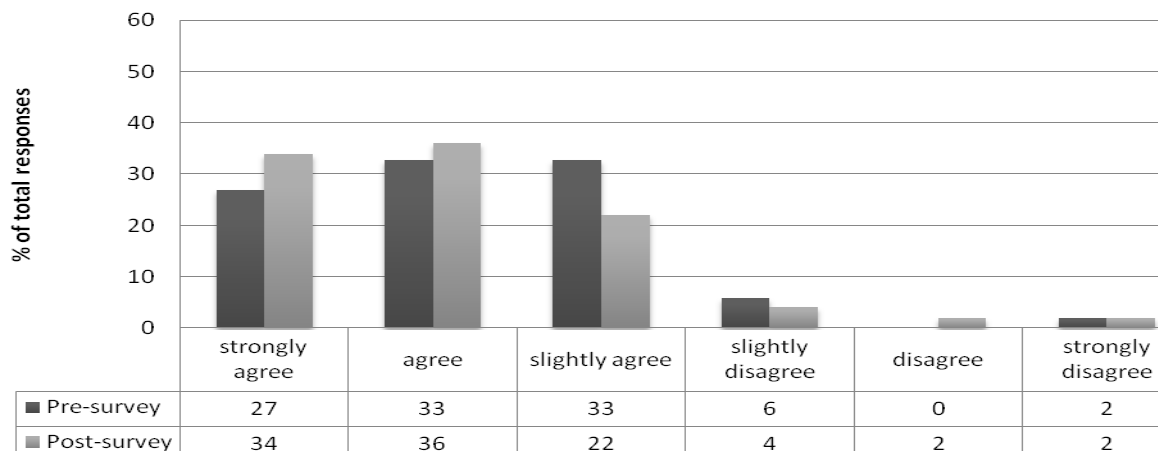


Figure 2. Response to "Version control software will reduce duplicated effort within my team."

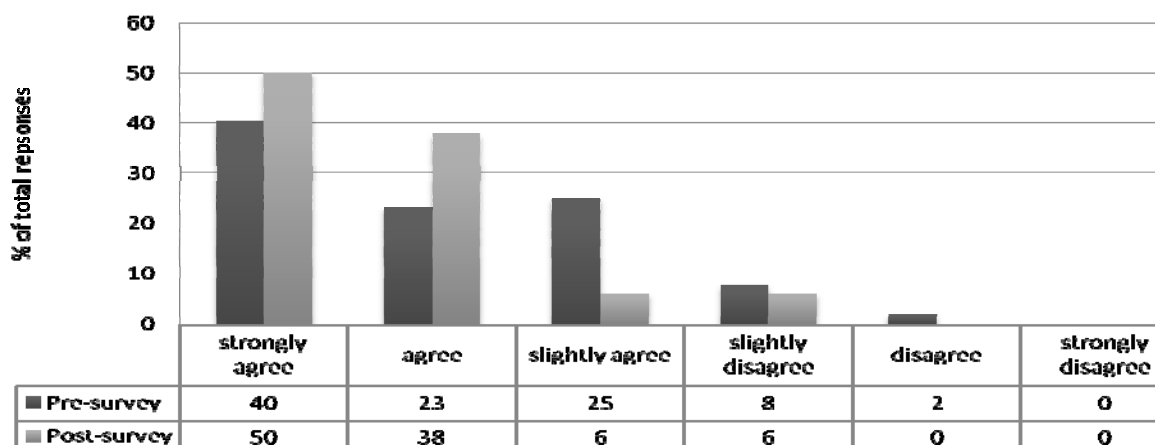


Figure 3. Response to "Version control software will enable the members of my team to obtain the correct versions of each others' code reliably."

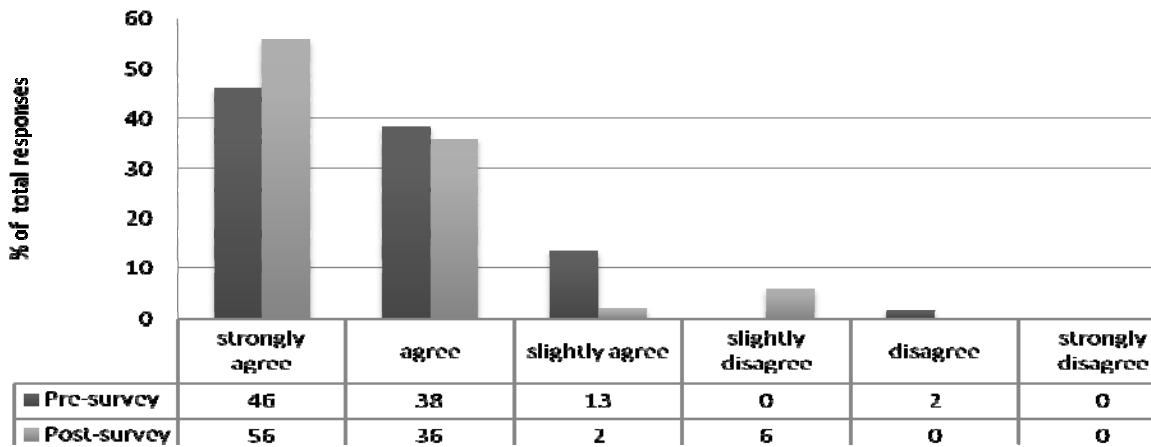


Figure 4. Response to “Version control software will enable me to share code with my teammates efficiently. ”

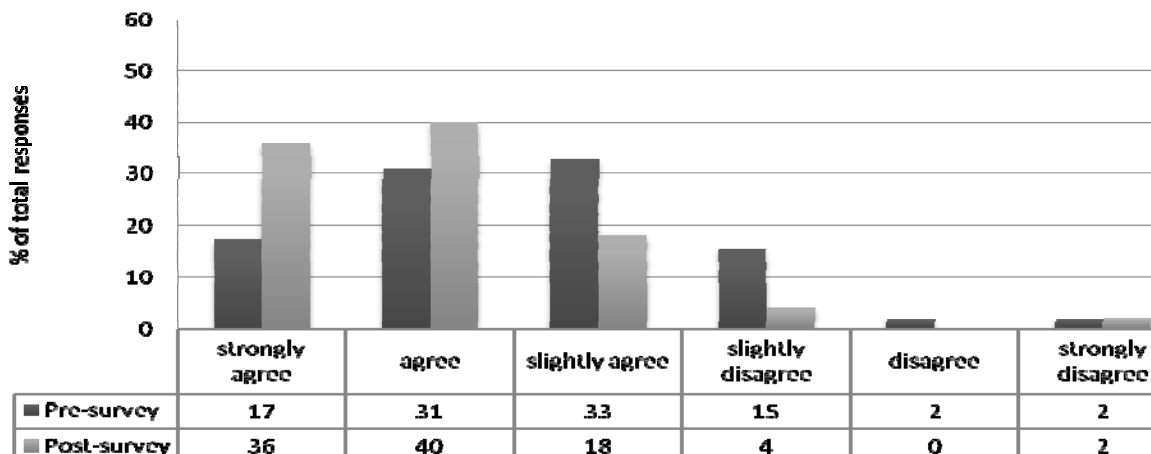
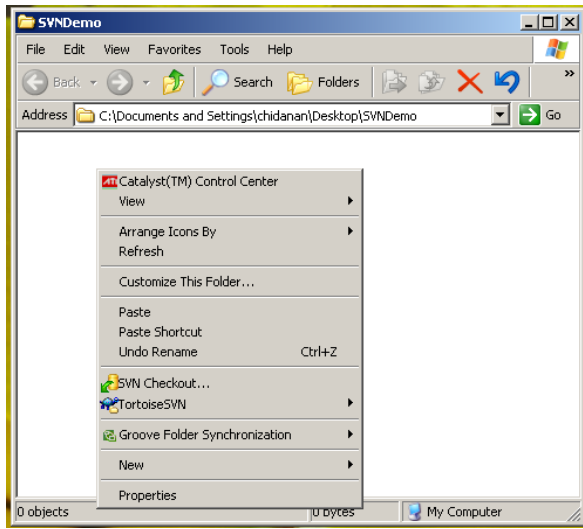


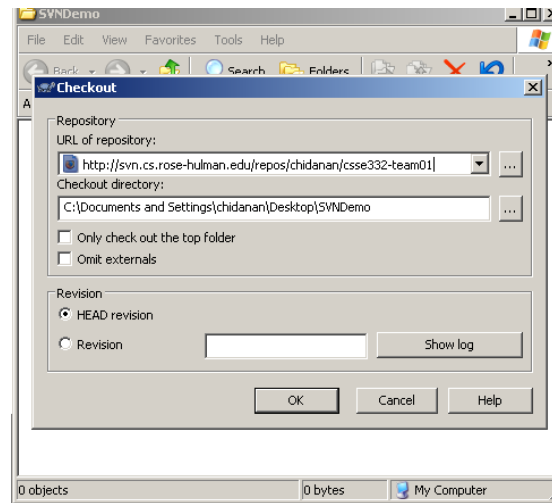
Figure 5. Response to "Version control software will enable my instructor to answer my code-related questions efficiently and effectively."

```
[chidanan@abacus ~/SVNDemo]$ svn co http://svn.csse.rose-hulman.edu/repos/chidanan/csse232-team01
A   csse232-team01/Instructions.txt
A   csse232-team01/test1.txt
Checked out revision 35.
[chidanan@abacus ~/SVNDemo]$ cd csse232-team01/
[chidanan@abacus csse232-team01]$ cp ../test2.txt test2.txt
[chidanan@abacus csse232-team01]$ ls
Instructions.txt test1.txt test2.txt
[chidanan@abacus csse232-team01]$ svn add test2.txt
A   test2.txt
[chidanan@abacus csse232-team01]$ svn commit -m "Checking in file test2.txt" .
Adding      test2.txt
Transmitting file data .
Committed revision 36.
[chidanan@abacus csse232-team01]$ ls
Instructions.txt test1.txt test2.txt
[chidanan@abacus csse232-team01]$ svn ls
Instructions.txt
test1.txt
[chidanan@abacus csse232-team01]$ █
```

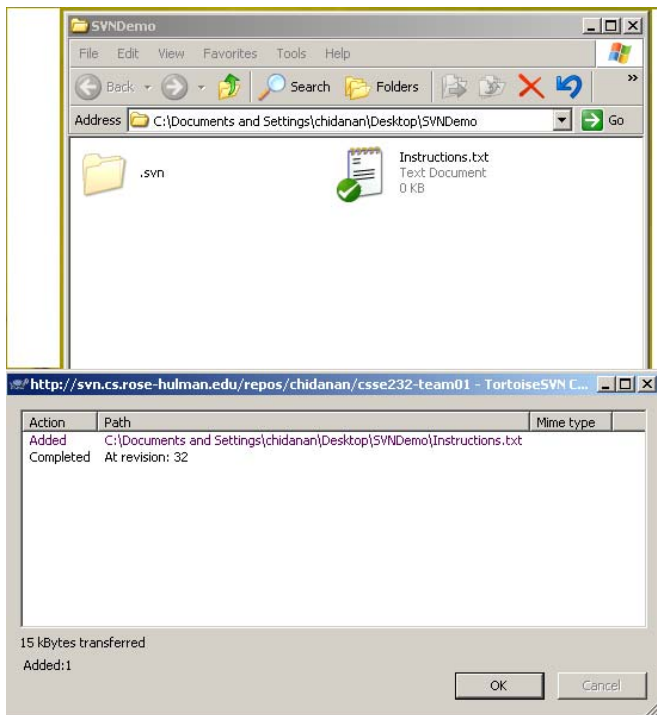
Figure 6. Steps to checkout and update a repository’s contents from the command line.



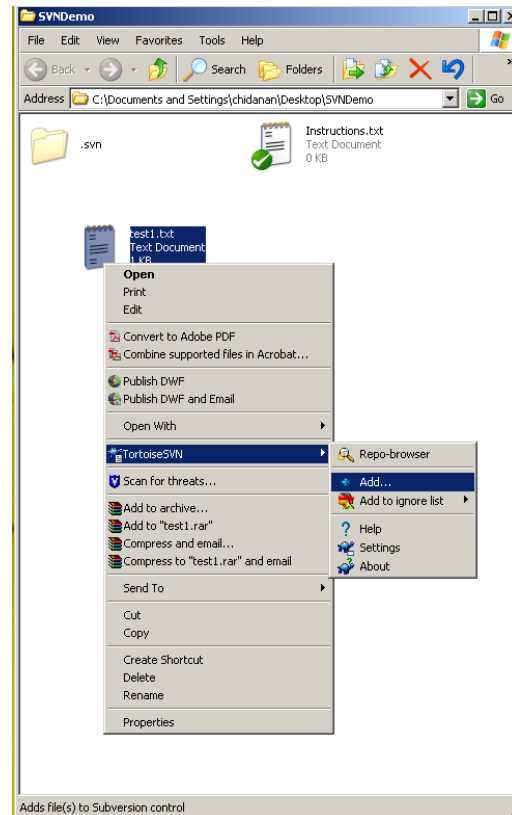
(a) Right-click to checkout from the repository.



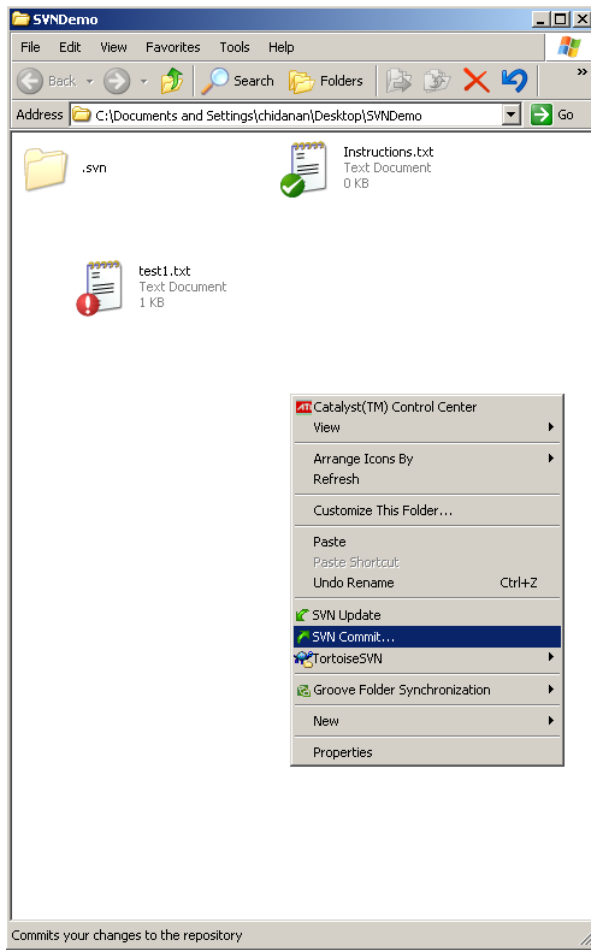
(b) Specify location of repository.



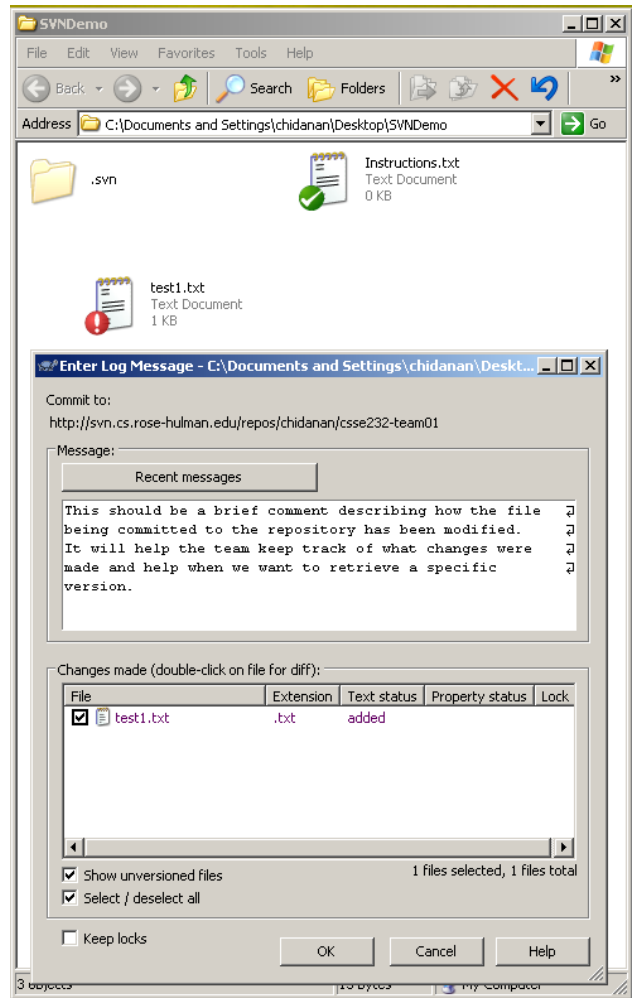
(c) Contents of repository have been copied to local folder.



(d) Adding a new file to the repository.



(e) Committing the modified file to the repository.



(f) Adding a meaningful message and selecting files to commit.

Figure 7. Steps to checkout and update a repository's contents using the Tortoise SVN GUI.

To install an SVN server on a local Windows machine:

Step 1: Install a server such as Apache:

- a. Login as localmgr or administrator and download and install the msi file from [12].

Step 2: Install Subversion:

- b. Download and install the msi file from [13].

Step 3: Create a repository:

- c. At the command prompt (Start->Run->cmd), type the following to create a folder called /svn to hold all the repositories.

```
prompt>>mkdir c:\svn
```

- d. Next create a repository called "team01" in the c:\svn folder.

```
prompt>>svnadmin create c:\svn\team01
```

Step 4: Configure Apache to run the subversion server:

- e. Edit the Apache configuration file, by following the Windows menus:

Start/All Programs/Apache HTTP Server 2.0/Configure Apache Server/Edit the Apache httpd.conf Configuration File

Uncomment the following statements in the modules section (i.e. remove the # before the statement):

```
LoadModule dav_fs_module modules/mod_dav_fs.so
```

```
LoadModule dav_module modules/mod_dav.so
```


Add the following statement to the modules section:

```
LoadModule dav_svn_module modules/mod_dav_svn.so
```

Add the following statements at the end of the file:

```
<Location /svn/team01>  
    DAV svn  
    SVNPath c:\svn\team01  
</Location>
```

Step 5: Start the subversion server and access the repository:

- f. Restart the server from the Monitor Services icon () on the taskbar.
- g. Open a browser and type the following address: <http://localhost/svn/team01>
A message similar to this should appear:

team01 - Revision 0: /

Powered by [Subversion](#) version 1.5.1 (r32289).

Step 6: Secure your server and allow access to specific users:

- h. Create a passwords folder c:\passwd and create a user called "username1" with a password:

At the command prompt, type the following:

```
prompt>> mkdir c:\passwd
```

```
prompt>>"c:\Program Files\Apache Group\Apache2\bin\htpasswd.exe" -c c:\passwd\passwords username1
```

Automatically using MD5 format.

New password: *****

Re-type new password: *****

Adding password for user username1

- i. In the Apache configuration file, modify the Location directive for the "team01" repository by adding the following lines:

```
AuthType Basic  
AuthName "Subversion Repository"  
AuthUserFile c:\passwd\passwords  
Require valid-user
```

where c:\passwd\passwords is the location of the passwords file for the "team01" repository.

- j. Now, if you were to open the browser and access the webpage again, you would be prompted for the username and password.

Step 7: Accessing the server and repository contents from another computer:

- k. If you want to access the contents of the repository from another computer, then you have to know the IP address of your computer (i.e. the one hosting the SVN server.). Go to the command prompt of the computer on which the SVN server is running and type the following:

```
prompt>>ipconfig
```

Depending on whether you are connected wirelessly or wired, look under Ethernet adapter Wireless Network Connection or Ethernet adapter Local Area Connection for the IP address (e.g. 74.125.67.100).

- l. Open a browser on the other computer and type: <http://IP-address/svn/team01> i.e.
<http://74.125.67.100/svn/team01>

You now have an SVN server up and running. For individuals to access the repository and make changes to its contents (add files, modify files and so on), you are encouraged to install a SVN client GUI such as TortoiseSVN(for Windows) [7] or RapidSVN(for Linux) [9].

Figure 8. Instructions to setup an SVN server on a local Windows machine.