A Temporal File System for Student's Assignments in The System Svetovid

IVAN PRIBELA and DONI PRACNER, University of Novi Sad

Storing students' assignments in practical computer science exams has several specific problems. In this paper we present some of the advantages of using a file system with a temporal dimension and present two working solutions, one of which was used in several courses with hundreds of students.

Categories and Subject Descriptors: K.3.2 [**Computers And Education**]: Computer and Information Science Education—*Computer science education*; D.2.11 [**Software Engineering**]: Software Architectures—*Data abstraction*

General Terms: Design, Experimentation

Additional Key Words and Phrases: file system, temporal, git, submission system, Svetovid

1. INTRODUCTION

Many computer science courses are focused on practical exercises as a form of continual assessment and examination. This approach undoubtedly offers friendly conditions to the students. As they use computers throughout the process of program development, this conditions are very close to the environment they will meet at work. However, in such circumstances, there are a few major problems that come from intricacies of electronic submission of the final solutions.

The first problem stems from the tendency of students to cheat and collude during solution development. This is only amplified by poor academic discipline among the students [Pribela et al. 2009].

Another problem is posed by bad or incomplete submissions on the part of the students, whether intentional or unintentional [Reek 1989]. If a student forgets to submit some of the required files the program might fail during assessment. Contacting the student to get the missing files delays the assessment, and there is no guarantee that the student has not been working on the missing part after the due date had passed.

The last problem is that in most cases only the final solution is submitted. There is no record of the effort or the path that the student took to come to the submitted state [Joy et al. 2005]. After the submission it is too late for any corrections, unless the teacher is willing to accept re-submissions and has spare time to re-evaluate them.

The described problems come from our personal experience, but are also recognised by many other institutions [Reek 1989; Hawkes 1998; Dempster 1998; Joy et al. 2005; Vujošević-Janičić et al. 2013; Christian and Trivedi 2016].

This work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. III 47003: Infrastructure for technology enhanced learning in Serbia;
Author's address: Ivan Pribela, Doni Pracner, University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia; email: ivan.pribela@dmi.uns.ac.rs doni.pracner@dmi.uns.ac.rs

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Belgrade, Serbia, 11-13.9.2017, Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

To solve them, a better method of submission is needed. This method should allow a student to give a complete and usable copy of the solution to the teacher without making it accessible to other students. Also the teacher should be able to run and check the solution without effort and focus on giving prompt feedback to the student.

As most of submission systems are focused only on submission, we have developed an interactive development environment integrated with a secure submission system in one package [Pribela et al. 2009].

This environment, however, focuses mainly on the prevention of plagiarism and collusion among the students. It provides to the students a standard set of tools expected in development environments like access to a console, compilation and execution of the code, etc. It also incorporates an editor specially designed to enable copying, pasting, dragging and dropping within the environment but disallowing it from and to the outside. This environment represents the first layer of defense against cheating: the prevention.

Described features are not the only vectors that students use to cheat, with prevailing other channel being the retyping from other sources. With this in mind and in the light of ever evolving and ubiquitous social media platforms, a second layer of defense is recommended. This second layer represent detection and investigation of any wrongdoing and is enabled by tracking of history and evolution of the student's solution through time. Although detection can never be completely certain the added information can only help in the investigation.

To address this and the other problems, we implemented a special file system that is described in this paper.

Usually, different students use different tools and programming environments while working on the solutions to the programming assignments. This diversity enables every student to use their preferred tools and environments. When the student is done working on the solution, the submission procedure is started. The usual submission process then copies the necessary files and records the submission timestamp.

To address this, we envisioned a process of continual submission. More precisely, a snapshot of the code is taken and submitted at key points in time during the work on the assignment. All snapshots are stored in a single file system structure and the project state can be recalled as it was at previous moments in time.

The requirements for this special file system are outlined in the following paragraphs.

The Svetovid system should manage multiple hierarchical file systems. These systems will be used to store and retrieve data like texts of student assignments, students' solutions of these assignments, test suites for the submitted solutions, or any other kind of files.

The requirements placed for the individual file systems are very similar to requirements for many other modern file systems. Each file system has a root folder which contains other folders and files. These file systems work just like any other hierarchical file system, files have content, while folders hold files and other subfolders.

However, unlike most other modern file systems, a detailed history of meta data and file content is kept. This means that any file system can be viewed as it was at any moment in the past. This enables the teacher to examine how the student's solution evolved, or to check if there were any inconsistencies and the student might be cheating. The student can also view all previous versions of their own files giving them the possibility to revert to an older version if that is more satisfactory.

The rest of this paper is organised as follows: Section 2 present existing solutions for storing data in file systems, mainly virtual file systems and version control systems. Section 3 describes the two solutions given, one with a general temporal implementation (Section 3.1) and the other that relies

on the Git version control system (Section 3.2. Finally a general overview, conclusions and options for future work are given in Section 4.

2. EXISTING SOLUTIONS

2.1 Virtual File Systems

In implementing the proposed solution, we took the approach of a Virtual File System. A virtual file system provides an abstraction layer on top of a more concrete file system. The main goal of this approach is to provide access to different types of concrete file systems in a uniform way. As we also wanted to support multiple platforms, using Java was a natural choice as it is platform independent.

When it comes to Java programming language and its libraries, there are a few good virtual file system interfaces that are available.

Apache Commons Virtual File System (Commons VFS) is one of them. It is a single consistent API for accessing numerous file system types and has a lot of mature features like caching, event delivery, integration support, etc [The Apache Software Foundation 2016a].

Eclipse File System (EFS) is another solution that is used mainly to store data in the Eclipse workspace [Eclipse Foundation 2017]. It is an API in the Eclipse platform to abstract details of the concrete file system. Although open and freely extensible it is not widely used outside the Eclipse environment, as if forces file system implementations to fit into narrow concepts for the Eclipse Workspace [Blewitt 2006].

Besides these two libraries, Java NIO.2 File System was released with Java 1.7 as a standard interface with attempt to modernise and improve native Java support for filesystem access. It is comparable in features with the previous libraries, but its late arrival is the main cause for its slow adoption [Leonard 2012].

2.2 Version Control Systems

All previously mentioned file system interfaces are missing the ability to keep track of changes in the data and meta-data. To address this, we looked at version control systems (VCSs).

A version control system is a system that records changes to one or multiple files and can recall specific versions. Although the main usage comes from versioning of software source code it can be applied to nearly any type of files.

The first VCS implementations were local version control systems that usually kept a simple database with all the changes. One such example is RCS [Free Software Foundation 2015].

To facilitate developer collaboration on projects, centralized version control systems like CVS [Vesperman 2007; Free Software Foundation 2016], Subversion [Fitzpatrick et al. 2008; The Apache Software Foundation 2016b] and Perforce [Wingerd 2006] were implemented. These systems feature a single server that contains all the versioned files, and a number of clients that check out and commit files. Recently, this centralized systems started to be replaced by distributed version control systems [de Alwis and Sillito 2009; Muşlu et al. 2014] mainly because the central server provided a single point of failure for the whole system.

In distributed VCS like Git [Chacon and Straub 2014], Mercurial [O'Sullivan 2009] or Bazaar [Gyerik 2013]) clients fully mirror the whole repository instead of just checking out latest snapshot of few files. These systems allow developers to work offline or during periods when the central server is unavailable. Furthermore, in case of the server crash every client has the complete repository and the data can be easily recovered. Also many of these systems can handle several remote repositories and enable developers to simultaneously collaborate with different groups of people in different ways within the same project.

2.2.1 *Git.* Git is a popular free and open source version control system designed for use by software developers. Although it is excells at managing changes to software source code, it can track any kind of content. It is designed to handle both small and large projects very effectively and offers fast performance and flexible workflows [Chacon and Straub 2014].

Git was created by Linus Torvalds to support the development of the Linux kernel. It replaced the previous version control system, BitKeeper, in managing the Linux kernel source code. As BitKeeper provided sophisticated operations that did not exist in other similar software available at the time, Git had to satisfy all the requirements and outperform the competition.

The new system allowed both independent and simultaneous development in private repositories as well as parallel development. The system enabled this without the need for constant synchronization with a central repository, as this could become a development bottleneck. Most of all, the tool allowed multiple developers in multiple locations to work in parallel even if some of them are temporarily offline. Each Git repository holds a complete copy of all the data. In order to achieve fast local updates and network transfer operations, Git uses compression and delta techniques to reduce transfer times and save space.

Git is using a cryptographic hash function to identify objects. This guarantees data integrity and allows trust to be established between all distributed repositories. Data objects stored in the Git repository database cannot be modified once created. As a consequence, the entire history of stored changes cannot be altered. Git also ensures that there is an accountability trail for every change by recording who comitted which content.

Git also supports the branching and merging workflow, like most other VCSs, but also goes a step further in promoting and encouraging it mainly by ease of creating new and merging old branches. Being free and open, Git was quickly adopted by the developer community and many other tools were built on top of original functionality.

3. IMPLEMENTATIONS IN SVETOVID

The File Systems service is implemented in Java as an OSGi service, which brings many benefits. It allows other OSGi services in the same container to communicate with no efficiency penalty, as the communication is realised through normal method calls. Also, the service can be imported in other OSGi containers using OSGi Remote Services Specification. Furthermore, the same service can be consumed as a web service by other services and end consumers that can be implemented in any programming language and deployed on any platform.

3.1 Temporal File System

One of the main ideas in the project was to have a temporal dimension for almost all the data in the whole system, not just in the file system. Several interfaces and classes were developed as OSGi bundles to enable this, starting with Period (Figure 1) that represents a time period with a first and last instant, both of which are long values compliant with Unix time, that is the number of miliseconds since midnight January 1, 1970, UTC. An expansion of this interface is the ValuePeriod that attaches a value to a period. The default implementations for these interfaces are immutable since objects of these classes will be passed around quite offten and need to guarantee the information they represent has not changed.

These first classes could be viewed as “atoms” in this systems that are used by the more complex classes. The interface TemporalValue defines multiple states of a variable, attached to different periods of time, and during some of them it can even be in a “not set” state, thus representing the change of a single value through time. Similarly there are TemporalSet and TemporalMap interfaces that work in the usual way and represent a set and a map that changes through time. To enable the persistence of the

```
package org.svetovid.core.util.temporal;

public interface Period {

    public long getFirstInstant();
    public long getLastInstant();

    public Period extendOver(Period period);
    public Period intersectWith(Period period);

}
```

Fig. 1. Interface Period

```
package org.svetovid.core.util.temporal;

public interface TemporalValue<T> {

    public boolean isEmpty();
    public boolean hasValue(Period... period);

    public Set<T> getValue(Period... period);
    public List<ValuePeriod<T>> getValuePeriods(Period... periods);
    public List<Period> getPeriods(T value, Period... periods);

    public void setValue(T value, Period... periods);
    public void setValue(ValuePeriod<? extends T> valuePeriod);
    public void addValue(TemporalValue<? extends T> value);

    public void clearValue(Period... periods);
    public void retainValue(Period... periods);

    .....

}
```

Fig. 2. Interface TemporalValue

data another OSGi bundle was made that connects these temporal concepts with the *Java Persistence API (JPA)*. For instance it features a JPATemporalMap class that is then used in many other classes as the main data storage with key-value pairs.

The implementation of the file system that is discussed here uses this temporal persisted map to store all the properties related to the files it needs to handle. Multiple versions of the files are stored in different root folders. The root folder for a particular moment in time is stored as a property. This way the teacher can define the root folder to change on every new assignment, or on predefined intervals, or even on every save, while at the same time the folder contents at any point is available to the compiler or any other external tool that needs to be run.

3.2 File System with a Git Back End

However, while implementing the mentioned solution some problem were encountered. Namely, any tools and environments that are needed or wanted had to support the new file system. In other words we had to integrate them and implement the needed glue code. This had to be done for each tool in order to support it so adoption was hampered. Because of this a different solution was tried.

Attributing to excellent features and abundant availability of numerous implementations and tools, Git was adopted as a foundation for our file system. Moreover it fulfils all laid out requirements and

its accessibility greatly helps consumers of the service consequently ensuring that it can be employed without difficulties. The only open issue is the choice and administration of the repository server that will be hosting the file systems.

Diverse open and free solutions are always welcomed, but every system brings slightly different concepts and ideas about the design and in our situation more importantly management. To offset this issue an abstraction layer was implemented on top of these servers to reconcile this diversity. This layer enabled our system to work with any type of Git repository. The operations themselves (such as saving a file) are implemented with additional calls to the adequate Git operations. The current version doesn't fully implement the whole temporal file system interface. It does allow for reverting to older versions of the files, but these are saved as separate operations, therefore the complete history is perserved for future analysis.

3.3 Fused File Systems

The files owned by a student are contained in a virtual file system (Figure 3) that is accessible only from that student's account. This virtual file system is a sum of three concrete file systems, located on the server. One file system is reserved for each student; content of which can be freely changed by the student. Figure 3 (b) shows examples of such virtual file system. Every assignment is also designated to one concrete file system whose files are accessible by students solving the same assignment. This is meant to contain description of the assignment, test data, and other files needed by students working on the assignment. On top of that, there is an additional file system, shared by all the students. The information necessary for all students is stored here.

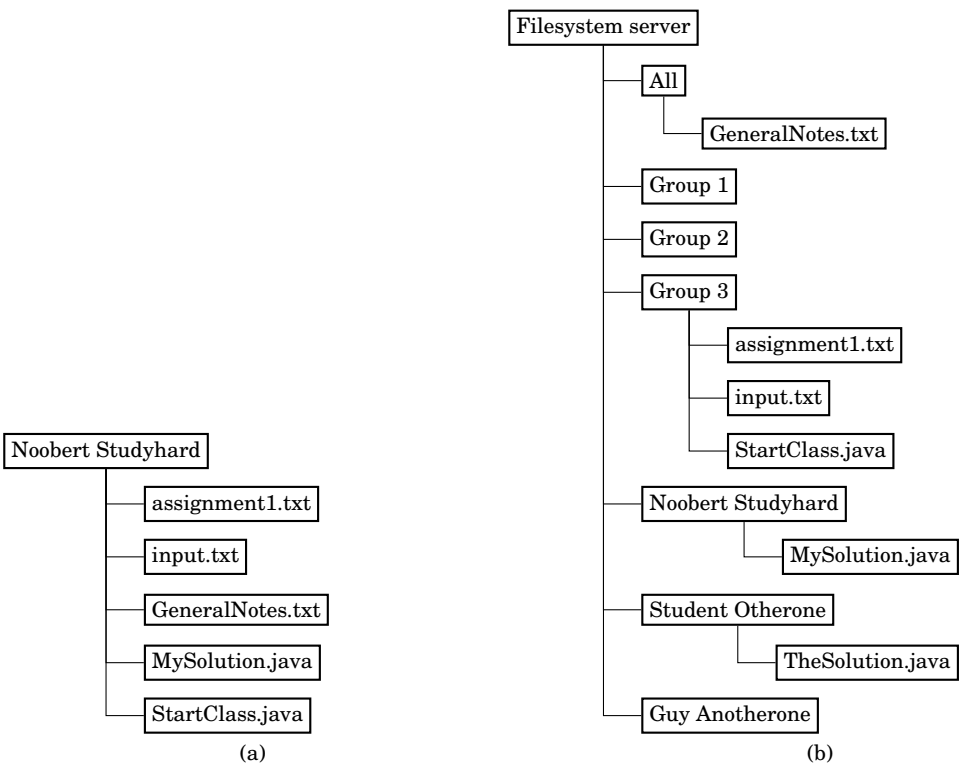

Fig. 3. Virtual directory as perceived by student (a), and as stored on server (b)

4. CONCLUSIONS

There are many real world practical problems with practical computer assignments, both with the collection of the solutions and with their grading. In general specialised tools are desirable for many of these tasks. In this paper we present some advantages of having a file system for storing this data that has a temporal dimension. This means that not only the final submitted version is stored, which leads to several advantages. The student could restore an older version of the assignment that was possibly more correct. The teacher can also have a better understanding of the development process of the solution and can evaluate its evolution and not only the final state.

Also, if any irregularities are detected there is the advantage of having a better overview of the whole proccess by which the student come to the submitted solution. For example, it can be seen whether a student tried many iterations of the solution, each building on the previous and eliminating some errors and implementing more and more of the assignment, or a student just wrote the whole solution and submitted it without ever trying it. However, the detection of cheating itself, whether automatic or maual, is not the responsibility of the filesystem and is left to other tools. The filesystem provides the necessary information to help in the further investigation if there is a suspicion.

The paper also presents two working solutions, one of which has been used in several courses with hundreds of students.

The presented solutions are generic file systems, meaning they are not meant only to keep students' assignments. However these systems store historic data about the properties and contents of the files, unlike most "normal" file systems, and offer the ability to review or restore previous versions of files. The first implementation (Section 3.1) uses custom built OSGi bundles with temporal properties and Java Persistence API for long term data storage. The other implementation (Section 3.2) uses the widely used version control system Git for data storage. This has the advantage of being easily combined with other tools and environments, since the individual file systems can be exported as regular Git repositories.

Although there is positive feedback for the first temporal file system solution, it is hard to measure the impact on student cheating rates or teacher's effort. There were a few informal interviews with our colleagues who stated an improvement, and there are plans to conduct a formal survey among both students and teachers in order to have a detailed description of their experiences and how they perceive the advances in all the challenges introduced in this system.

There is always room for improvements with any essential service such as a file system. Both presented solutions have their advantages and disadvantages. Many of them would be solved if the temporal file system would be implemented with a new bundle supported by the Git version control system. There are also options to integrate the file system to be directly usable by Eclipse IDE for courses that require the advanced features that an IDE can provide. Similar could be said for other such programs.

Specific tools could be developed that would use the history saved by this system to point out possible irregularities and give a more in depth view of the evolution of the students' solutions.

REFERENCES

Alex Blewitt. 2006. Eclipse File System. http://www.eclipsezone.com/articles/efs/. (2006). Accessed: 2017-07-01.

Scott Chacon and Ben Straub. 2014. *Pro Git* (2nd ed.). Apress, Berkely, CA, USA.

Maxwell Christian and Bhushan Trivedi. 2016. A Comparison of Existing Tools for Evaluation of Programming Exercises. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (ICTCS '16)*. ACM, New York, NY, USA, Article 134, 6 pages. DOI: http://dx.doi.org/10.1145/2905055.2905350

Brian de Alwis and Jonathan Sillito. 2009. Why Are Software Projects Moving from Centralized to Decentralized Version Control Systems?. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE '09)*. IEEE Computer Society, Washington, DC, USA, 36–39. DOI: http://dx.doi.org/10.1109/CHASE.2009.5071408

Jay Dempster. 1998. Web-based assessment software: Fit for purpose or squeeze to fit. *Computer Assisted Assessment* 6, Article 1 (1998).

Eclipse Foundation. 2017. The Eclipse File System. https://wiki.eclipse.org/EFS. (2017). Accessed: 2017-07-01.

Brian Fitzpatrick, C. Michael Pilato, and Ben Collins-Sussman. 2008. *Version Control with Subversion: Next Generation Open Source Version Control*. O'Reilly Media, Farnham, UK.

Free Software Foundation. 2015. GNU RCS Manual. https://www.gnu.org/software/rcs/manual/. (2015). Accessed: 2017-07-01.

Free Software Foundation. 2016. Version Management with CVS. https://web.archive.org/web/20140709051732/http://ximbiot.com/cvs/manual/. (2016). Accessed: 2017-07-01.

Janos Gyerik. 2013. *Bazaar Version Control*. Packt Publishing, Birmingham, UK.

Trevor Hawkes. 1998. An experiment in computer-assisted assessment. *Computer Assisted Assessment* 6 (1998).

Mike Joy, Nathan Griffiths, and Russell Boyatt. 2005. The boss online submission and assessment system. *Journal on Educational Resources in Computing (JERIC)* 5, 3, Article 2 (Sept. 2005). DOI: http://dx.doi.org/10.1145/1163405.1163407

Anghel Leonard. 2012. *Pro Java 7 NIO.2*. Apress, New York, NY, USA. DOI: http://dx.doi.org/10.1007/978-1-4302-4012-9

Kıvanç Muşlu, Christian Bird, Nachiappan Nagappan, and Jacek Czerwonka. 2014. Transition from Centralized to Decentralized Version Control Systems: A Case Study on Reasons, Barriers, and Outcomes. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 334–344. DOI: http://dx.doi.org/10.1145/2568225.2568284

Bryan O'Sullivan. 2009. *Mercurial: The Definitive Guide*. O'Reilly Media, Farnham, UK.

Ivan Pribela, Mirjana Ivanović, and Zoran Budimac. 2009. Svetovid - Interactive development and submission system with prevention of academic collusion in computer programming. *British Journal of Educational Technology* 40, 6 (Nov. 2009), 1076–1093. DOI: http://dx.doi.org/10.1111/j.1467-8535.2008.00894.x

Kenneth Reek. 1989. The TRY System -or- How to Avoid Testing Student Programs. *SIGCSE Bulletin* 21, 1 (Feb. 1989), 112–116. DOI: http://dx.doi.org/10.1145/65294.71198

The Apache Software Foundation. 2016a. Apache Commons Virtual File System. https://commons.apache.org/proper/commons-vfs/. (2016). Accessed: 2017-07-01.

The Apache Software Foundation. 2016b. Apache Subversion Documentation. https://subversion.apache.org/docs/. (2016). Accessed: 2017-07-01.

Jennifer Vesperman. 2007. *Essential CVS*. O'Reilly Media, Farnham, UK.

Milena Vujošević-Janičić, Mladen Nikolić, Dušan Tošić, and Viktor Kuncak. 2013. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Information and Software Technology* 55, 6 (June 2013), 1004–1016. DOI: http://dx.doi.org/10.1016/j.infsof.2012.12.005

Laura Wingerd. 2006. *Practical Perforce*. O'Reilly Media, Farnham, UK.