Accumulative Versioning File System Moraine and Its Application to Metrics Environment MAME

Tetsuo Yamamoto
Graduate School of Engineering Science
Osaka University
t-yamamt@ics.es.osaka-u.ac.jp

Makoto Matsushita
Graduate School of Engineering Science
Osaka University
matusita@ics.es.osaka-u.ac.jp

Katsuro Inoue
Graduate School of Engineering Science
Osaka University,
Graduate School of Information Science
Nara Institute of Science and Technology
inoue@ics.es.osaka-u.ac.jp

ABSTRACT

It is essential to manage versions of software products created during software development. There are various versioning tools actually used in these days, although most of them require the developers to issue management commands for consistent versioning. In this paper, we present a novel versioning file system Moraine, which accumulatively and automatically collects all files created or modified. Those files are versioned and stored as compressed forms. The older versions are easily retrieved from Moraine by the time-stamps or tags if required.

Using Moraine system, we have developed a metrics (measurement) environment called MAME (Moraine As a Metrics Environment). MAME can collect various metrics data for on-going or past projects, since its basis, Moraine, is able to retrieve all versions of all products (files).

Both Moraine and MAME have been implemented. Using these systems, we have evaluated the performance of Moraine and MAME with various test data and student project data. The result shows that disk space required by this approach is several times larger than ordinary approaches; however, it is acceptable at the current tendency of disk price decrease. By this approach, an ideal metrics environment has been easily established by developing simple data-collection tools for version files.

1. INTRODUCTION

Software systems are becoming large and complex, and managing software development projects are getting hard and difficult. One of important issues in the management works is controlling product versions.

SIGSOFT 2000 (FSE-8) 11/00 San Diego, CA, USA

In many software development organizations, there are specific managers who are responsible to the product configuration and versions. Through the configuration manager, the proper versions of products are kept in the storage, and past versions are retrieved from it in consistent manner. For this purpose, configuration management tools RCS[21] and CVS[4] have been widely used. These tools are very convenient; however, we have to properly learn their proprietary configuration models to keep consistent versions. Also, we need to issue configuration commands at proper timing.

In this paper, we present a versioning file system named Moraine[23] which is designated to support evolution activities in software development. Moraine accumulatively records the history of all files; i.e., every file once created is preserved and each update to the files is recorded in the file system. The user of Moraine does not need to worry about the management commands for versioning activities such as check-in and check-out. The system automatically records an updated file as the newest version associated with old versions, and it provides the newest version as a current file. Number of versions are stored as compressed forms using the difference of two subsequent versions. Stored versions are retrieved by the version numbers, time stamps, or user-defined tags.

The file system in VMS[14] operating system records file changes automatically. VMS saves all of the contents of newly saved file, and these saved contents are identified by a special suffix of filename. Our file system differ from VMS file system in several respects. Our file system is that the actual version management part is separated from the file system part.

Through several experiments, we are confident that Moraine is very practical to various software development projects under current tendency of drastic decrease of disk price and increase of CPU power. It is very practical to assume that each software developer can use large amount of disk space with high-speed CPU.

Moraine system alone works fine for managing versions of software products. In this paper, we also consider an application of Moraine to software metrics environment. Software Metrics are extending its importance to software project success. In these days, many software development projects in the large use quality frameworks such as CMM[18], ISO–9000[8], and SPICE[9]. In these

quality frameworks, quantitative data collection is essential as the bases of project evaluation and management. For projects in the small, importance of quantitive data is also emphasized in the context of individual capability improvement, such as Personal Software Process[7].

It is very important to plan the collection strategy before projects start. For example, GQM paradigm[2] requires setting the goals for project evaluation and identifying approaches to the goals as questions. After determining the goals and questions, the metrics to be collected are selected. This top down approach for data collection would work fine if we are able to the design project structure well before the project starts. We can initiate metrics data collection from the beginning of the project.

However, there would be a lot of cases where we would notice the need of new software metrics data during project running or even after project termination. In such cases, it is difficult in general to get the metrics data for the past period of the project. Collecting data for deleted products and terminated processes is infeasible in currently available software development environments.

Moraine, which keeps fine-grain versions of all files, equips very good features as a software metrics environment. The Moraine system automatically stores all versions of all files without extra human management. We can restore any version of any file if it once existed in the system. Therefore, we are able to start collecting metrics data even after the project initiation.

In this paper, we show a metrics environment named MAME (Moraine As a Metrics Environment), as an application of Moraine. Since Moraine is implemented as a transparent file system, we can easily built MAME above Moraine.

There are a lot of metrics environments proposed and actually implemented[3, 13, 20, 22]. However, most of those are proprietary environments, where a pre-designed metrics collection policy is to be determined. For MAME, on the other hand, we can determine the policy even after project initiation.

MAME has been actually applied to a student project of complier construction. Several metrics data have been collected after the project, and the characteristics of the student activities have been detected. Although this experiment is limited in the sense of program size and development period, we would think that MAME is very applicable to various fields requiring software metrics.

Contributions of this paper are proposal of MAME, a new architecture for metrics environment, as well as its base system Moraine, an accumulative file versioning file system. Also, we will present practicability of this approach through the experiments.

This paper is organized as follows. In Section 2, we briefly describe the overview of an accumulative versioning file system Moraine. Section 3 shows evaluation results of Moraine. Section 4 presents the approach of using Moraine as a metrics environment MAME. An experiment of applying MAME to the student project is shown in Section 5, and we discuss on the approach of Moraine and MAME in Section 6. Section 7 concludes our discussions with several remarks.

2. OVERVIEW OF VERSIONING FILE SYSTEM MORAINE

In this section, we present Moraine, which is an accumulative software development environment based on a versioning file system.

2.1 Design Policy

As we notice day after day, the price of hard disks is getting lower and lower, and the power of consumer-level CPU is getting higher and higher. We would hope a development environment such that all of the developers activities to the development environment are recorded to a storage, and that any past data could be retrieved easily from the storage if needed.

Such development environment records all versions of all files we have created. We do not need to care about preserving versions of files, and we can delete files from the environment if it is currently needless. The deleted files are retrieved by specifying time stamps or configuration tags given by the user.

Based on these observations, we have established the following design policies of Moraine.

- Easy operation: the users are not required to learn how to use Moraine. Usual file read/write operations are automatically hooked, and versioning works are performed by Moraine without user's hand.
- Open structure: Moraine should not have proprietary structure of data repository or versioning tools, and it is easily ported to many systems.

Based on these concepts, we have designed the system as a single virtual file system of UNIX environment with system call hook mechanisms. With this approach, the users are not necessary to recognize the versioning operations; they simply issue file read and write system calls to the kernel. In other words, they develop a software without the concept of repository and workspace. Also since the system is designed as a virtual file system, we can easily separate the implementation issues from this architecture, and can port to other machine environments.

2.2 Architecture of Moraine

Figure 1 shows the architecture of Moraine we propose. The core part of the system is called VCFS (Version Control File System), which contains several components, VFS (Virtual File System), VCD (Version Control Daemon), RCS (Revision Control System)[21] as an available versioning sub-system, and several control command tools.

In VCFS, operations to a file (read and write) are automatically mapped into activities of version management; engineers do not consider what should be done to manage the product versions. There are no difference between the operations to usual file system and VCFS file system from a viewpoint of users' processes.

VCFS manages the versions of regular files (symbolic link, special file, socket, and named-pipe are out of our scope). A new version of a file is created and checked-in fully automatically iff a file is created or an existing file is changed. Checking-out the latest version is done with simply reading the file. VCFS also supports a file locking mechanism. Before a check-in operation is completed, other processes can only check-out the file.

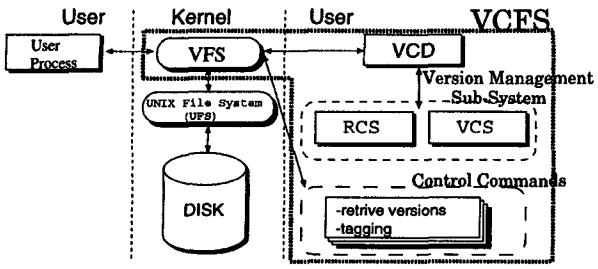

Figure 1: Architecture of Moraine

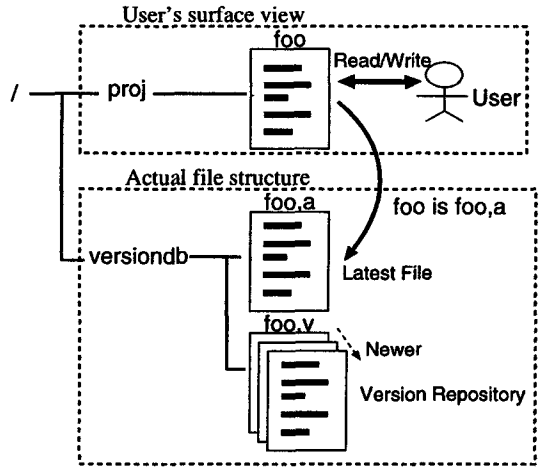

Figure 2: Mapping files between VCFS and actual file system

VCFS internally employs "check-in/check-out model"[5] which was used by RCS; however, it is not mandatory. We can change the models if needed.

VCFS is implemented as a stackable file system through VFS component, i.e., all files handled by VCFS are stored not directly to the UNIX raw file system, but indirectly via VFS which write finally to the UNIX file system.

Each user file consists of two actual files; one is the latest version, and another is the version repository (Figure 2).

Usually VCFS shows the latest version of the file to the user. For example, we assume that a file system */versiondb* is mounted to */proj* by VCFS. Now the user creates a file */proj/foo*. VCFS creates *"/versiondb/foo,a"* as the latest version of file *foo* (*/proj/foo* itself) and *"/versiondb/foo,v"* (if RCS is used as a version management sub-system), for the version repository. Note that */versiondb/foo,v* is invisible from the user surface with ordinary operations (we need version management operations to get older versions).

VCFS always keeps the latest version of every file (*/versiondb/foo,a* in the previous example) for fast file read/write to the latest version. When a UNIX process opens a file in read-only mode, VCFS behaves as the same as NULL file system[19](which simply passes the operations to the raw file system without modification). When a process opens a file in write-only or read-write mode, VCFS hooks close() system call and performs a check-in operation to a file. The check-in operation is activated by VCD.

A stackable file system doesn't do a change inside the existent file system. Moreover, the output of the file system is done as a file in file system which becomes the lower layer, and it isn't done directly by hard disks and so on. It doesn't need to write the code which it deals with a storage device and a network with, and is a portability.

VCD is a daemon process which acts as a bridge between the kernel and the version management sub-system. VCD dispatches the requests from the kernel to the version management sub-system.

The version management sub-system is the actual version management part of VCFS. In general, version management sub-system consists of a set of tools. VCFS employs external version management systems as the sub-systems, and we can change the subsystems to use. Current prototype of VCFS has two kinds of version management sub-systems, RCS and VCS. VCS is a simple version management system. VCS saves all versions as-is, and does not calculate the delta between versions. No version derivation is allowed, however, registering a new version is faster than the RCS sub-system.

Control command tools help to control system behavior. Examples of commands include, retrieving previous versions, making a branch, showing a delta between versions, and so on.

Current prototype of Moraine runs on FreeBSD 3.0-RELEASE[6], a BSD UNIX[12, 15] variants. VCFS is written in C and about 5000 lines in total.

3. EVALUATION OF MORAINE

In this section, we discusses Moraine from viewpoints of the system performance and stored data size. It is important to know that the system has acceptable performance for software development environment. Therefore we have measured the performance of the file system about reading and writing files.

We use UNIX file system (denoted UFS) and NULL file system (NULLFS) to compare with our Moraine (VCFS). All experiments were made on a machine with 166MHz Pentium CPU and 48MB RAM running FreeBSD 3.0-RELEASE.

3.1 Performance of File Read and Write

At first, we measured an elapsed time for a UNIX process to read different 1MB files repeatedly. "An elapsed time" means time between process initiation and process termination. Be aware that the elapsed time includes an overhead of typical UNIX processes (process initialization, etc).

Figure 3 shows the results of the reading test of VCFS, UFS, and NULLFS. "VCFS" represents VCFS using RCS as the version management sub-system. The vertical axis shows the elapsed time, and the horizontal axis shows the number of read operations.

This graph shows that VCFS and NULLFS take almost the same time, and they are a little slower than UFS. This is because both VCFS and NULLFS contain the overhead for the implementation of the stackable file system, which slightly reduces the system performance. However, the difference is small and limited.

We also measured elapsed time to write files, similar to the file reading test described above. Figure 4 shows the results of writing test. "VCFS+" represents VCFS using VCS as the version management sub-system. "VCFS+" indicates the elapsed time including

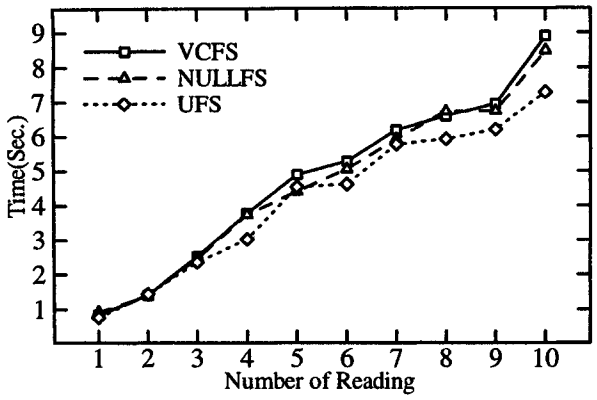

Figure 3: Performance of File Read

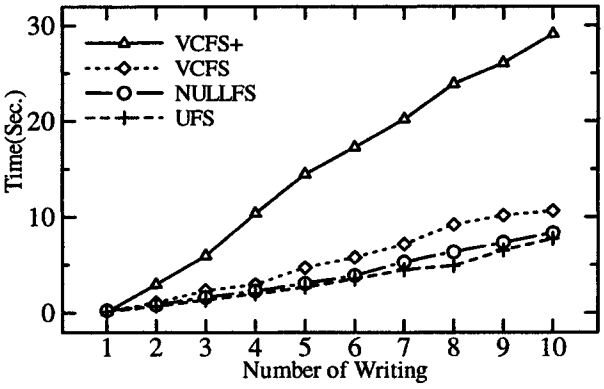

Figure 4: Performance of File Write

time for synchronization between VCD and RCS (i.e., waiting the completion of the file write operations). In the case of VCFS, we do not wait for the termination of RCS.

NULLFS is about 10% slower than UFS, very similar to the reading test. Writing a file in VCFS requires a new version registration overhead which is not required by the reading; thus VCFS consumes 30% or more extra time compared with UFS.

VCFS+ is several times slower than UFS, NULLFS, and VCFS. This is because VCFS+ count for time of computing file difference. However, this time consuming process is usually performed as a background job and the users do not need to wait for it. The users can get prompt from the system with the waiting time of VCFS, not VCFS+.

Finally, a sequence of file read and write based on practical setting has been performed. We have compiled several UNIX application tools using provided "make" files. Two applications, "tar" and "dump" bundled with FreeBSD have been compiled on VCFS (Moraine), NULLFS, and UFS. Table 1 shows the total lines of source codes and the total number of source files of those applications. The results of elapsed time for completing "make" programs of those applications are shown in Table 2.

The procedure of compiling an application includes reading operations of the source codes and writing operations of the object codes repeatedly. As a result, VCFS is about 20% slower than UFS; however we consider that this overhead is not a serious problem as a practical software development environment.

Table 1: Data of Applications

	Total lines of codes	Total number of files
dump	15123	25
tar	3705	9

Table 2: Compile an Application (Sec.)

	dump	tar
VCFS	12.79	33.46
NULLFS	11.3	32.01
UFS	10.9	28.27

3.2 Stored Data Size

We applied Moraine to the student project of Osaka University, where each student develops a compiler for Pascal-like language in C. Table 3 shows various characteristics of the development. The total lines of codes shows the number of all lines in the C source files with the latest version numbers. The total number of files is the number of distinct files without counting the versions. The total number of versions is the number of all versions for all files including C source files, object files, and others. The value in the parenthesis is the number of versions for C source files only. For example, student 2 created finely 4067 lines of C code, and he made 20 different files which have 249 versions all together including 147 versions of C source files.

Table 4 shows disk space usages under UFS and VCFS. (NULLFS is the same as UFS.) Note that all object files and executable files once created are also stored in the disk.

The results of UFS show actual sizes of final versions of products. The disk space requirements for VCFS are 6–8 times greater than those of UFS. However, it is only 1.4 MBytes even for the largest case, which is fairly small amount under huge disk space currently available with moderate prices.

4. USING MORAINE AS A METRICS ENVIRONMENT

In this section, we introduce some requirements for metrics environment which supports development activities driven with quantitive measurement. Metrics environment is an infrastructure for quantitive process and/or product measurement of ordinary software development activities.

Metrics environments collect some quantitive (not subjective) metrics data from activities of engineers. The environments also store the data, and provide facilities to analyse the data. Also, metrics environments are used as a back-end for software process assessment tools.

4.1 Requirements to Metrics Environments

We think there are four essential requirements of metrics environments as follows:

Table 3: Characteristics of Project

	Total lines of codes	Total number of files	Total number of versions (source files)
student1	9339	45	533 (311)
student2	4067	20	249 (147)
student3	2543	18	357 (247)

Table 4: Total Disk Space Usage (K bytes)

	UFS	VCFS
student1	225	1388
student2	117	546
student3	73	604

1. *No extra burden should be imposed to the developers.* It is not pleasant that the developer is forced to do special activities for data collection such as "please use our special tool", "please do pre-defined activity, and do not do others". We have to consider psychological overhead of those activities. Moreover, imposing such activities to the developer would cause potentially problems of the quality of the collected data.

2. *Various kinds of metrics data should be easily collected.* We might built a tool collection composed of a bit of small tools. However, we would have more *generic infrastructure* to collect *lots of* metrics; we would like "environment" (rather than a tool-set).

3. *Various granularity data should be easily collected.* Software development activities should have lots of aspects, such as engineers, managers, and so on. Since we can consider various granularity levels to a single metrics, environments should provide a facility to show from detailed to abstracted view of the metrics.

4. *Data format of collected metrics should be no proprietary and widely applicable.* Recently, open-source movement[11] is wide-spreaded. Software architecture, design, and implementation which used to be hidden, become open to all of us. In such situations, using a commonly-used data format to store collected metrics data is very important requirement for metrics environment. It brings a chance to enhance the metrics environments.

4.2 Design of MAME Architecture

Figure 5 shows the architecture of MAME (Moraine As a Metrics Environment). Development environment is put on Moraine, and all accesses to files within the development environment are processed by Moraine. Engineers can also retrieve the results of development activities from Moraine with metrics tools such as data presentation tool, data analyzation, etc.

Note that we do not need to change the development environment. It is simply on Moraine, and the developer is not necessary to to be aware re of MAME.

Various kinds of metrics tools can be included in MAME. Current implementation provides several size metrics tools written in Perl. We easily combine these metrics data. These tools provide also the number of versions, a modified date of a version, the author of a version and a file list of the directory.

Figure 6 shows a screen image of Version Viewer, accessed through

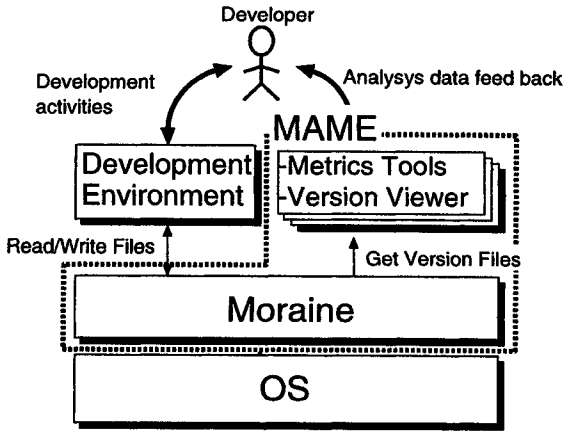

Figure 5: MAME architecture

a Web browser. The left-most column of this table shows the version numbers from the initial version 1.1 to the latest version. A row of each version consists of the version number, the date of the file update, the author name who updateded the file, the number of added/deleted lines, a bar graph of the number of lines, and optional tag for identifying this version. A change of file can be viewed visually using this tool.

4.3 Features of MAME

MAME has sufficient features to cover the requirements of metrics environments. The following are major features of MAME assorted by the requirements mentioned above.

1. In MAME, engineers can use their own environment as they used. MAME does not change engineers working environment. MAME co-exists those environments and works as metrics environment. Also, engineers can get information of the current status of the development from MAME.

2. MAME collects data of most activities on the environment. We assumes that most activities of software development are summarized as operations of files. MAME collects all file operations such as changing file contents, and creating a new file. These operations includes various types of activities of software development.

3. MAME provides fine-grain data and can be abstracted. The data which is collected by MAME are the results of primitive operations. It can be used as fine-grain data without any processing. Also, these data can be considered as a series of file operation. We would abstract the the data of MAME to be able to fit high-level software development models.

4. MAME does not employ a proprietary data format. MAME uses other external tools to record data, and MAME does not

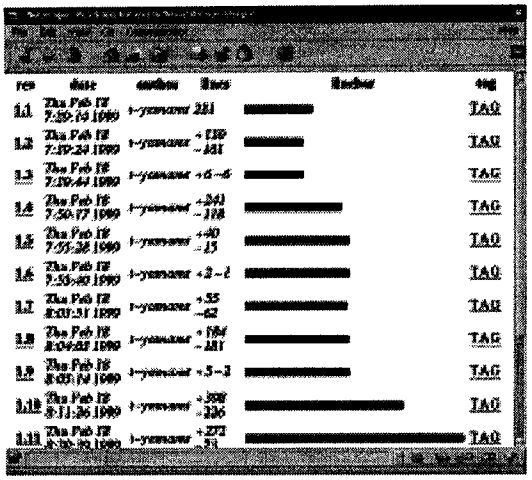

Figure 6: Version Viewer

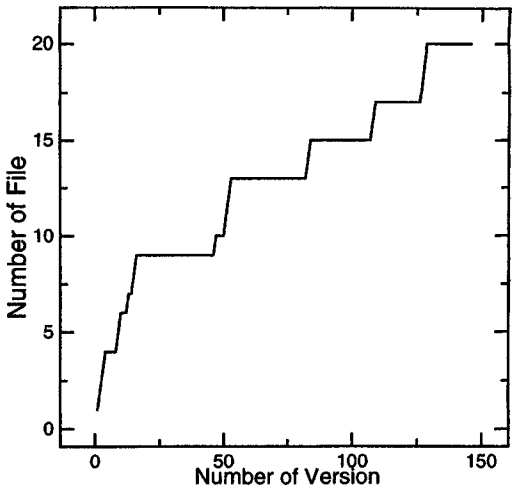

Figure 7: Changes of Number of Files (Student 2)

have its own proprietary data format. We can use familiar data manipulation tool to process MAME data.

5. EXPERIMENT OF USING MAME

5.1 Overview of Experiment

We have applied MAME to the same student project of the compiler construction presented in Section 3. In Table 3, we had shown the total lines of codes of the last versions, the total number of files, the total number of created versions (including the number of C source files).

The data collection was performed after the project termination. MAME extended the compressed diff files to the original forms, and computed several metrics values. Here, we computed three metrics to know the behaviour of the students during the project. In this experiment, we have collected the number of files (exsisting at that time), total lines of codes in the files, and the number of C functions in the program files.

Figure 7, Figure 8, and Figure 9 show those metrics values for student 2, respectively. The horizontal axis for each graph is the cumulative number of versions for C source files(totally 147 versions). This metrics is used here as the elapsed time of the student project. This value would be better for the student than the calender time, since students usually work sporadically and the calender time would be no use.

5.2 Interpretations of Metrics Data

From Figure 7 and Figure 8, we could guess that the project progressed fairly smoothly without serious troubles, since the graphs almost grow monotonically. Between version number 20 – 50, there are no increase of all graphs. This would indicate the student mostly did minor modifications in the files.

Figure 9 shows a rapid increase at very early stage (around version number 20) and following long stable period. This would show that the student first created the skeleton of necessary functions, then he filled the contents of the functions. Also, the student added many functions at late stage (around version number 100) of the development.

These observations have been easily obtained from the output of MAME.

6. DISCUSSIONS

6.1 Moraine

The architecture of Moraine is quite unique one as a versioning tools for software development. All created versions of all files are automatically collected by the hidden process, and the developers are not necessary identify the versioning mechanisms or commands.

As a versioning tool, the features of Moraine is sufficient in the sense that we can store and retrieve any versions of files effectively. Adding explicit tags to versions would ease the searching efforts of specific versions.

Moraine does not have configuration management mechanisms as CVS[4] and RCS[21] have. A configuration of a set of files have to be managed by hand, or by creating configuration files. However, it is not difficult to develop, as an application on Moraine, a configuration management environment on Moraine.

Current implementation of Moraine provides retrieval features based on time-stamps or annotated tags. These are almost the same as CVS and RCS features. However, the granularity of the versions stored in Moraine is generally finer than those in CVS and RCS. Therefore, we would need more sophisticated mechanisms to search a past version out of fairly large number of versions.

Moraine collects automatically and accumulatively all versions of all files as a non proprietary system. Although there is no same system as Moraine, there are related commercial tools such as ClearCase[1], PVCS[16] and Visual Source Safe[17]. These tools can collect versions of objects in various granularity levels. However, the engineers have to recognize the system and to issue check-in/check-out commands for the version management. Also there is a tool such as 3-D file system[10]. 3-D Filesystem is implemented as a modified file system of UNIX System V release 3. Merging version management features to native file system makes a tool-free operation for extracting any version; however, registering a new version requires

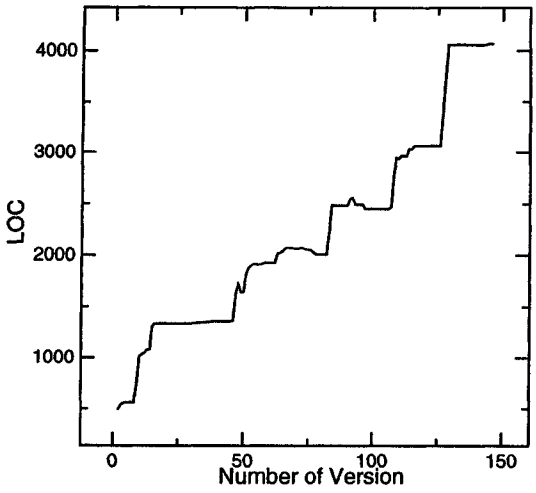

Figure 8: Changes of Lines of Code (Student 2)

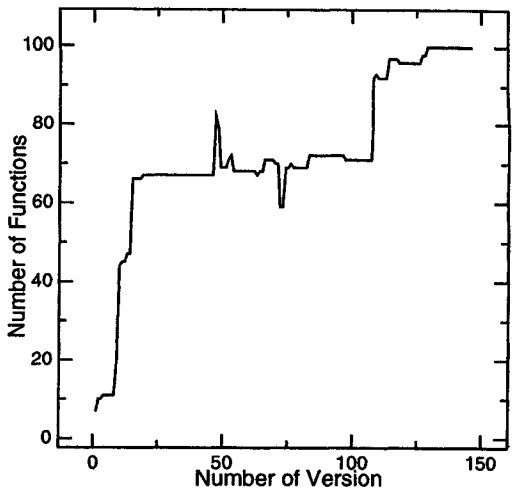

Figure 9: Changes of Number of Functions (Student 2)

some supporting tool associated with the file system. On the other hand, Moraine fully automates the version store operations through file write operations. This would be a substantial difference .

Performance of Moraine is acceptable as a base of software development platforms. The performance decrease by keeping every version is limited. Although the system load factor would be increased by computing file difference, this computation job is performed in the background and we do not need to wait the computation.

Moraine would be able to be a foundation of new software development environment models. Current software development environments require a lot of management operations on file, product, or object names. We have to keep all necessary files in the system, and this is fairly complicated and troublesome works. Using Moraine, we can delete files which are currently unnecessary. The files which might be needed in the future can delete from the surface of the system (but Moraine keeps it). By using versions retrieval features in the version space, we would recover the necessary file to the surface of the system.

6.2 MAME

MAME has been constructed as an application of Moraine, a fine-grain versioning file system. It has a novel architecture for software metrics environment. MAME system would meet the requirements for metrics environment shown in Section 4. There are no specific enforcement to the developers. The management overhead for collecting metrics data is very limited, where all versions of all files are automatically preserved in the system. Those versions are restored when we gather metrics data. Various tools for computing various metrics data are executed for the restored version files. The system is fairly transparent since the data store is created as a virtual file system, and the metrics tools and viewer are located beside the interaction between the developer and the development environment.

The extreme nature of MAME is that we can gather metrics data for past versions or deleted files. Thus, we can set up or change data collection policy during the project or after the project. Ordinary metrics environments require predetermined data collection policy. This nature is established by recording all versions once created in the system. The experiment data shown in Section 3 and Section 5 indicate practicability of this approach. The disk space requirement is greater than the usual file systems but still affordable.

By using MAME, we can gather product data fairly easily, although performing the process data collection would require more sophisticated ways. With current implementation of MAME, we would analyze the command history files for getting process data, or we would construct process data from the product data which are fully available by Moraine. The latter approach will work fine in the cases that we can easily guess the developer's activities from the version files. For example, the number of compilation would be guessed by the version number of the object files straightforward.

There are environments which can be used as metrics environments. METKIT (Metrics Education Toolkit)[20] can be used to collect metrics data in ordinary development environment. Engineers can introduce or create modules to collect data they want; it should be defined what metrics is used before it uses. Project Crocodile[13] is also metrics environment which integrates measurement tool sets and existing tool sets. However, it assumes that the criteria for the metrics is already defined before using the environment. TAME[3] is a project to establish a metrics environment. TAME introduces GQM paradigm[2] to collect metrics data, so it tends to be top down approach to select what metrics should be used before adapting to existing software development environment. Ginger2[22] also achieves to build a metrics environment. However, since its implementation employs propriety tool sets to collect data it is difficult to apply to actual environment.

7. CONCLUSION

In this paper, we presented Moraine, which supports various activities in software development. Moraine accumulatively records the history of all files. Moraine acts as a file system wrapper for existing version management system. We evaluated Moraine in the performance and file-size point of view. Moraine provides easy operations of the version management, open system structure. It is very practical to various software development projects.

We also proposed MAME, which is a metrics environment as an application of Moraine. MAME collects data of most activities on the environment without a burden to the developer. MAME was applied to the project, and various observations of the student activities has been easily obtained from the metrics data from MAME.

As a further work, we are planning to apply Moraine and MAME to an industrial-size project. In such case, the issue to be solved is not the disk space or CPU power of the development system, but maturity of the system. In order to ensure the reliability of the system, we are currently reviewing the implementation of Moraine, with directly interact with UNIX kernel.

8. REFERENCES

[1] Atria Software Inc. ClearCase product summary. Technical report, Atria Software Inc., 24 Prime park Way, Natick, Massachusetts 01760, 1994.

[2] V. R. Basili, G. Caldiera, and H. D. Rombach. Goal Question Metric Paradigm. In J. J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[3] V. R. Basili and H. D. Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[4] B. Berliner. CVS II: Parallelizing software development. In *Proceedings of 1990 Winter USENIX Conference*, Washington, D.C., Winter 1990.

[5] P. H. Feiler. Configuration management models in commercial environments. Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, Mar. 1991.

[6] J. K. Hubbard. RELEASE NOTES FreeBSD Release 3.0-RELEASE. "http://www.freebsd.org/releases/3.0R/notes.html".

[7] W. Humphrey. *Introduction to the Personal Software Process*. SEI Series in Software Engineering. Addison Wesley, 1997.

[8] International Organization for Standardization. *ISO 9000: Quality management and quality assurance standards; Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*. Geneva, Switzerland, 1991.

[9] ISO/IEC TR 15504. Software process assessment's parts 1–9, technical report type 2, 1998.

[10] D. G. Korn and E. Krell. A new dimension for the Unix file system. *Software–Practice and Experience*, 20(S1):19–34, June 1990.

[11] E. S. Laymond. The cathedral and the bazaar. "http://www.tuxedo.org/%7Eesr/writings/cathedral-bazaar/cathedral-bazaar.ps".

[12] S. Leffler, M. McKusick, M. karels, and J. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, 1989.

[13] C. Lewerents and S. F. A product metrics tool integrated into a software development environment. In *In Proc. European Software Measurement Conference FESMA98*, pages 603–608, 1998.

[14] K. McCoy. *VMS File System Internals*. Digital Press, 1990.

[15] M. McKusick, K. Bostic, M. karels, and J. Quarterman. *The Design and Implementation of the 4.4BSD UNIX Operating System*. Addison-Wesley, 1996.

[16] Merant, Inc. Merant pvcs version manager & configuration builder. "http://www.merant.com/products/pvcs/".

[17] Microsoft, Inc. Visual source safe. "http://msdn.microsoft.com/ssafe/".

[18] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):18–27, July 1993.

[19] J.-S. Pendry and M. McKusick. Union mounts in 4.4BSD-Lite. In *Proceedings of the USENIX 1995 Technical Conference*, pages 25–33, New Orleans, LA, USA, January 16–20 1995.

[20] M. Russell and M. Bush. Introduction to metkit. *Journal of Information and Software Technology*, 35(2):108–110, 1993.

[21] W. F. Tichy. RCS – a system for version control. *Software–Practice and Experience*, 15(7):637–654, July 1985.

[22] K. Torii, K. Matsumoto, K. Nakakoji, Y. Takada, S. Takada, and K. Shima. Ginger2: An environment for caese (computer-aided empirical software engineering). *IEEE Transactions on Software Engineering*, 25(4):474–492, 1999.

[23] T. Yamamoto, M. Matsushita, and K. Inoue. Moraine: An Accumulative Software Development Environment for Software Evolution. In *IWPSE99 International Workshop on the Principles of Software Evolution*, pages 89–93, Fukuoka, Japan, July 1999.