Journal of

Software Engineering

ISSN 1819-4311

Academic
Journals Inc.

www.academicjournals.com

Journal of Software Engineering 4 (3): 231-243, 2010
ISSN 1819-4311

Metrics Suite for Component Versioning Control Mechanism in Component-based Systems

Parminder Kaur and Hardeep Singh
Department of Computer Science and Engineering,
Guru Nanak Dev University, Amritsar-143005, India

Abstract: Version control is an important activity in the overall software development. With the focus shifting to component-based systems in recent times, use of version control techniques in component-based systems assumes a great importance. Hence, every version control system needs a close monitoring and control. Various open-source version control systems are analyzed and compared. It is felt that there is a need to have a generic framework, which can handle multiple versions of different types of components. This study presents a framework for managing configuration management issues related with component versioning mechanism and their implementation using developed Visual Version Control Tool (VVCT). A set of metrics for close monitoring and control of whole versioning system is also suggested.

Key words: Software components, version control, version control tools, component-based systems, component framework, visual version control tool

INTRODUCTION

Object-Oriented Design (OOD) frameworks serve as better components in software development as compared to the objects. In practical applications, objects assume to play more than one role in more than one context and it is possible only with respect to OOD frameworks (Coleman *et al*., 1994; Cook and Daniels, 1994; Mauth, 1996; Crnkovic *et al*., 1999). Frameworks allow components that play different roles in different frameworks to be composed by composing frameworks. Version Control (VC) systems help to manage multiple versions of the same component in a given environment (Thomas and Hunt, 2003; Collins-Sussman *et al*., 2006).

A typical component-based application is composed of several pre-existing components while some other components like application-specific components are developed by application developers. Every component used in a particular application evolves during time. Moreover, sometimes multiple different components provide similar functionality and out of them, some components can be substituted at a given place in the application architecture (Gergic, 2003; Larsson, 2000).

The developed version control tool, VVCT, considers a software component as a database or a text document, the different versions of which when created can be compared, reconstructed and deployed. For development of this tool VB.NET language is used as,

Corresponding Author: Parminder Kaur, Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar-143005, India
Tel: 0183-2258802-09/3573, 9914-010452 Fax: 0183-2258531

Microsoft's Visual Basic Framework is specialized for visual composition of components rather than for any particular runtime quality attribute. SQL Server 2005 is taken as a database repository, which stores the whole information related with different versions of the same component.

Mostly common available open-source version control systems, like CVS and Subversion (Thomas and Hunt, 2003; Collins-Sussman *et al*., 2006), work well with text documents only, whereas the tool developed, presently provide versioning support for two different types of components i.e. text document as well as database component. Any modification to these components, in the form of addition of new contents or deletion of existing contents, creates new version of the related component, which is then stored in the repository. The new versions as well as old versions can be assessed at any time for further use. This tool also supports the comparison of same database as well as text component irrespective of any two different variants of the same version.

A metric suite is suggested at four levels i.e., Time-based metrics, Personnel metrics, metrics at Framework level and metrics at Component level, in order to monitor and control the working of a versioning process with developed Visual Version Control Tool (VVCT).

Component Frameworks

Component framework refers to a framework, which defines a set of rules for different services, where evolution of new components takes place with new additions/modifications. Component frameworks are the basis of component-based software systems that define rules and guidelines so that components interact with each other and enable the modification of components at run-time. The life-cycle activities of software components and sharing of resources between several components are handled by component frameworks. Component frameworks allow components that play different roles in different frameworks to be composed by composing framework (Crnkovic *et al*., 1999).

The developed version control tool, VVCT, is implemented in .NET framework as. NET supports the multiple roles for a single component. .NET can use multiple interfaces for each role of a component. VVCT uses aggregation mechanism to compose new frameworks with the help of existing frameworks. Using aggregation, developed components can be reused for different purposes or to compose new frameworks. New component frameworks can be added at run-time by adding roles to a component.

Component Configuration Management Issues

Working with frameworks provides various advantages like management of various versions of same component but it also adds additional level of complexity (Larsson, 2000), (Pieber and Spoerk, 2008). Frameworks are considered as composite entities/components, where each entity/component has its own internal structure, which is built from components or from their parts. A framework component has relations with other frameworks and can also be composed from other frameworks. The creation of new component from existing component framework/frameworks introduces configuration management problems in following two cases:

- Managing same component in more than one frameworks
- Creating new framework from existing components and frameworks with respect to versions and variants

Case 1

Suppose there are two frameworks F_1 and F_2 along with relations R_{12} and R_{13}. Both frameworks share the component C_1:

$$F_1 = \{C_1, C_2: R_{12}\} \text{ and } F_2 = \{C_1, C_3: R_{13}\}$$

Let a new property be added into the component C_1, according to the new requirements of framework 2. It leads to the creation of new revision or variant of the component C_1 say $C_{1:V0.var0}$ into the framework 2 i.e.,

$$F_2 = \{C_{1:V0.var0}, C_3: R_{13}\} \quad \text{[where V0 ~ initial version and var0 ~ initial variant]}$$

To fulfill the emerging requirements, revisions can be performed on the same version of the component. As a result, different variants of the version like $C_{1:V0.var0}$..... $C_{1:V0.varn}$ are created in the repository. Any variant fulfilling the current requirements may be chosen and committed as a next version of the component e.g.,

$F_2 = \{C_{1:V1}, C_3: R_{13}\} = \{C_{1:V0.vark}, C_3: R_{13}\}$ [where k refers to any variant of the same version of the component]

In this situation, two possibilities can occur:

- Framework specification will remain the same if change in component version is not taken into consideration
- If change in component version considers with respect to the framework 1, then again two possibilities can occur i.e., either it works well or it may fails. As the current version of component C_1 is $C_{1:V1}$, framework automatically executes the latest variant i.e., $C_{1:V0.vark}$ of component C_1.

To avoid this unpredictable situation, basic configuration methods like version management of components and configuration management of frameworks can be used as follows (Crnkovic *et al*., 1999):

- A component should be identified by its name, version number and variant number
- A framework should also be identified by its name as well as version, so that new framework version can be derived from existing variants of component versions as well as by adding new components

The above said conditions suggest that new versions of frameworks can be configured at the time of creation of new component as follows:

$$F_{1:Vx} = \{C_{1:Vi.vark}, C_{2:Vj.var1}: R_{12}\} \quad F_{1:Vx+1} = \{C_{1:Vi+1.vark}, C_{2:Vj.var1}: R_{12}\}$$
$$F_{2:Vy} = \{C_{1:Vi.vark}, C_{3:Vk.var1}: R_{13}\} \quad F_{2:Vy+1} = \{C_{1:Vi+1.vark}, C_{3:Vk.var1}: R_{13}\}$$

This situation leads to a fact that more than one framework can share the different variants of same version of the component and one framework can share the same component with different versions along with their variants. If the other side is considered,

it increases the number of generated frameworks. Therefore, it becomes necessary to limit the number of desired configurations. This can be done by assembling the required versions of components in a form of baseline and derive the frameworks from baselined component versions. This concept is used by Crnkovic *et al.* (1999) and proved this fact that number of desired frameworks does not grow unnecessarily, if they are derived from baselined component versions.

When a new role is assigned to a component, then a need exists to change the specific part of the component. This change will affect only those frameworks, where that aspect is included whereas other frameworks though containing the component, do not affected with the change (an aspect is defined as a subset of a component and framework is defined as a set of aspect versions with relations between aspects). Therefore, it is better to keep the baselined versions/aspects under version control system. The following relations define this process:

$$C_{1:Vk} = \{ A_{j:Vl} \} \text{ where } Ai(C_k) \subseteq C_k \text{ and } F_{vk} = \{[A_{j:Vl}(C_{i:Vk})]: R_{jl}\}$$

Case 2

In OOD framework, it is possible to design a new framework. This new framework acts as a superset of classes and relations from the frameworks it is created. Component-based systems also support this fact that a new component-based framework can be developed with the help of existing frameworks along with newly added components. If it is created during run-time, then the components from the selected frameworks include the new framework. The proposed framework supports this fact. It creates a new framework by selecting the components from available frameworks with different versions.

Suppose F_1 and F_2 are two frameworks as follows:

$$F_1 = \{C_1, C_2 : R_{12}\} \quad F_2 = \{C_1, C_3 : R_{13}\}$$

And framework F_3 is obtained after combining these two frameworks as shown:

$$F_3 = \{C_1, C_2, C_3 : R_{12}, R_{13}, R_{23}\}$$

This composition works well as long as no change takes place in either framework. As new revisions occur due to changes in requirements, a new version with its variant, of the component C_1 is created in F_1 and keep the old version of the same component in F_2 then both frameworks can be defined as:

$$F_{1:Vi} = \{C_{1:Vi+1.vark}, C_{2:Vk}: R_{12}\}$$
$$F_{2:Vj} = \{C_{1:Vi.vark}, C_{3:Vk}: R_{13}\}$$

Two possible situations can occur while combining these two frameworks:

$$F_{3,Vl} = \{C_{1:Vi+1.vark}, C_{2:Vk}, C_{3,Vl} : R_{12}, R_{13}, R_{23}\}$$

And

$$F_{3,Vl} = \{C_{1:Vi.vark}, C_{1:Vi+1.vark}, C_{2:Vk}, C_{3,Vl} : R_{12}, R_{13}, R_{23}\}$$

The first framework $F_{3, V1}$ shows the creation of new framework with latest (new) version of component C_1 and second framework consist both versions of C_1 component. Thus, the proposed mathematical framework supports both possibilities and proves the fact that the new framework can include either latest version of the component or both (latest as well as previous) versions of the component. The support of handling more than one version also resolves various configuration management issues. This proves the fact that more than one versions of the same component can be handled by the developed framework, in the run-time environment.

Working of Visual Version Control Tool (VVCT)

Visual Version Control Tool (VVCT) refers to a framework, which can handle multiple versions of a component, where each version can have multiple variants. It provides all rules to guarantee good structuring of components and enough services to work properly.

Components handled by VVCT consist mainly of interfaces and implementation bodies. Server components provide their services through their interfaces, which act as contacts between client and server components. Components are stored in the form of a family where the term family refers to a logical organization of components and their interfaces, as shown in Fig. 1. Each component has its own version tree. Any version, along with any variant, can be accessed at any time. This framework also supports the merging of components from different versions and stores them as a new framework. This new framework contains components from older versions as well as newly added components and then stored in the repository with starting version say zero. Any modification in this version leads to the next version.

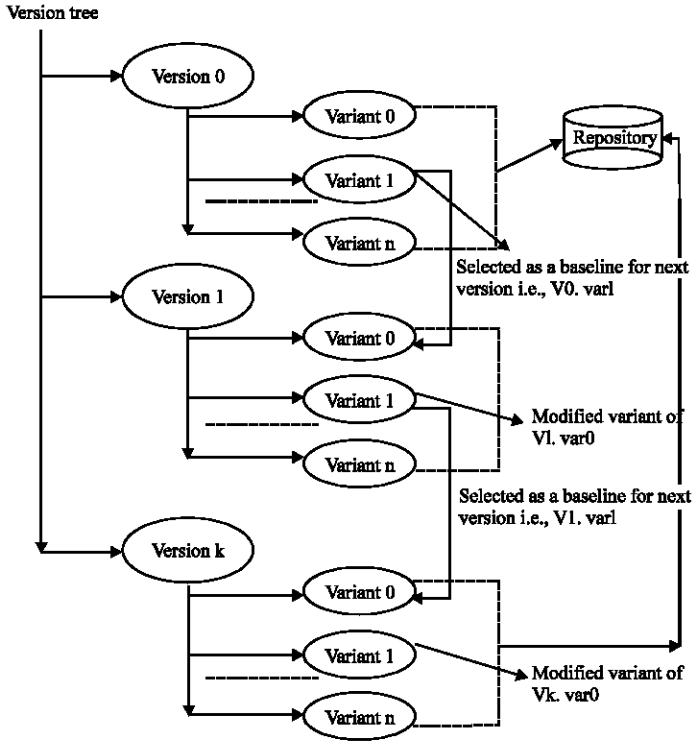

Fig. 1: Different versions of a component, along with their variants, in form of a version tree

VVCT also supports handling of multiple versions of text documents. Multiple versions can be accessed from the repository and comparisons can be performed. Differences between various versions of text files can also be compared.

VVCT has a repository manager, as shown in Fig. 1, which stores information with respect to components, their revisions and their interfaces. A component is indexed according to the interface it implements. A specific component, along with its version history, can be requested directly from the repository. Any variant of the existing version can become the baseline of the next version.

Few screen snapshots related with the working of VVCT are shown with the help of following figures. Initial screen consist the preview of two types of components, one is database component and other is text component as shown in Fig. 2 and 3.

VVCT also supports handling of multiple versions of text documents. Multiple versions can be accessed from the repository and comparisons can be performed. Differences between various versions of text files can also be compared.

VVCT has a repository manager, as shown in Fig. 1, which stores information with respect to components, their revisions and their interfaces. A component is indexed according to the interface it implements. A specific component, along with its version history, can be requested directly from the repository. Any variant of the existing version can become the baseline of the next version. Few screen snapshots related with the working of VVCT are shown with the help of following figures. Initial screen consist the preview of two types of components, one is database component and other is text component as shown in Fig. 2 and 3.

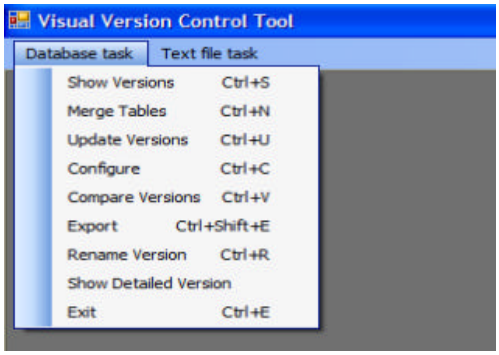

Fig. 2: Snapshot of various possible operations on database component

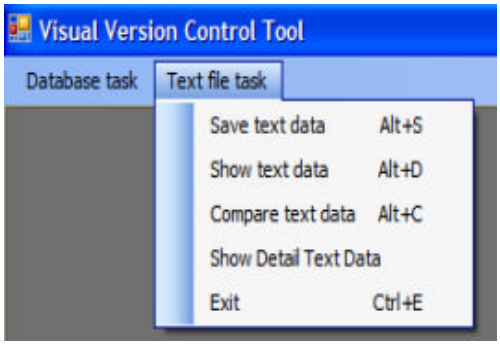

Fig. 3: Snapshot of Various Possible Operations on Text Component

Different operations that can be performed are as follows:

- Display of available variants of selected versions of components according to the specified path and their contents
- Comparison of different variants of same version of a component
- Developing a new framework by merging the components of different versions along with new additions (Fig. 4)
- Updating of versions after committing a change
- Display of available versions of text document according to the specified path and display of contents according to the selected version
- Compare different versions of a selected text file (as shown in Fig. 5)
- Creation of a log table in the form of text file with respect to the changes done in a text document as well as database structure

VVCT Versus other Version Control Tools

This section provides the comparison of VVCT, with different commercial as well as open-source version control tools with respect to various parameters. The several categories and sub-categories, considered for comparison, are discussed in Table 1. The detailed information with respect to these terms, as stated in Table 1, is available at http://better-scm.berlios.de/comparison/comparison.html.

Metrics Used For Version Control in VVCT

In order to monitor and control the working of a versioning process with Visual Version Control Tool (VVCT) and to implement the proposed model, the following set of metrics, divided in four major categories-time-based metrics, personnel metrics, metrics at component level and metrics at framework level, is suggested in its initial forms:

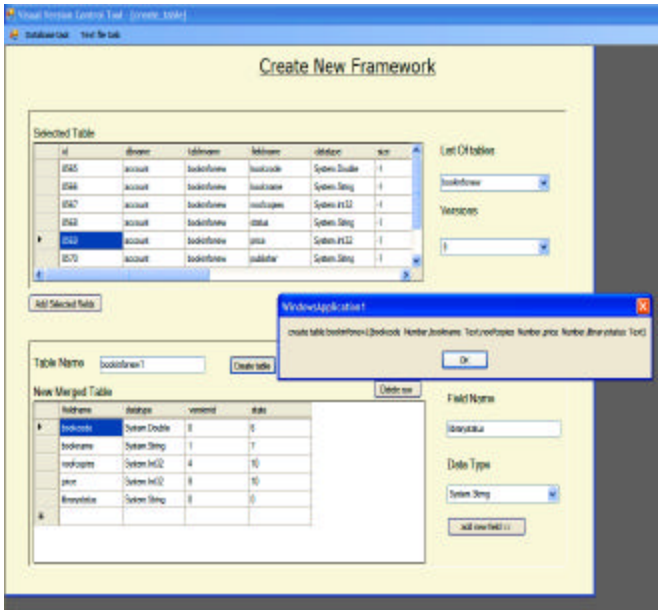

Fig. 4: Creation of New Framework with available versions and new additions

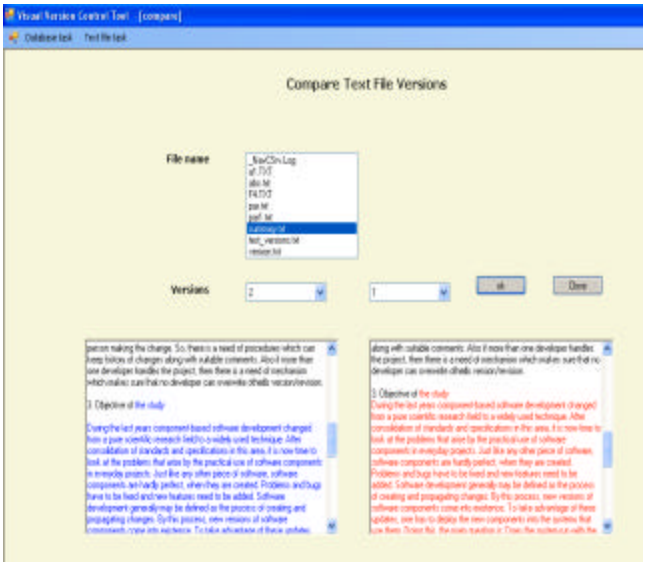

Fig. 5: Comparison of two versions of same text files after making new additions

Table 1: Comparison of VVCT with commercial as well as open-source version control systems

	Tools					
Categories	CVS	Subversion	Visual source safe	Perforce	Vesta	VVCT
Repository operations						
Atomic Commits	No	Yes	No	Yes	Yes	Yes
Intelligent Merging after Renames	No	Yes	No	Partial	Unknown	Yes
File and Directory Copies	No	Yes	Yes	Yes	Yes	No
Remote Repository Replication	Indirectly	Yes	Indirectly	Yes	Yes	Yes
Propagating Changes to Parent Repositories	No	Yes	Indirectly	No	Yes	No
Repository Permissions	Limited	Yes	Yes	Yes	Yes	Yes
Change sets Support	No	Partial	No	Yes	Not exactly	Yes
Tracking Line-wise history	Yes	Yes	Not Directly	Yes	No	Yes
Features						
Ability to work on one database/ directory of the repository	Partial	Partial	Partial	Partial	Partial	Yes
Tracking Uncommitted Changes	Yes	Yes	Yes	Yes	Yes	No
Per-file Commit Messages	No	No	Yes	No	Not exactly	Yes
Technical status						
Documentation	Excellent	Very Good	Medium	Very Good	Quite Thoroughly	Very Good
Ease of Deployment	Good	Good	Very Good	Very Good	Medium to Good	Very Good
Command Set	Simple	Easy	Little difficult	Very extensive	Little difficult	Not used
Networking Support	Good	Very Good	Good	Good	Very Good	Good
Portability	Good	Excellent	Good	Excellent	Good	Good
Automatic backup	Yes	Yes	Yes	Yes	Yes	Yes
User Interface						
Web Interface	Yes	Yes	No	Yes	Yes	No
Graphical User Interface	Very Good	Very Good	Standalone GUI	Very Good	Standalone Windows GUI	Very Good
Simplicity of Use	Many Commands complicated	Simple to use	Very simple	Simple	Simple	Very simple

Time-based Metrics

The time-based metrics suite consists following metrics:

- Total number of versions of a component i.e., Total Num Ver
- Total number of variants to a version i.e., Num Vari Ver
- Average number of variants to a version

$$\text{AvgNumVari\_ver} = \sum_{I=1}^{N} \text{NumVari\_ver/Total NumVer}$$

- Total time to generate all variants of same version i.e., Total Time Variant

$$\text{Total Time Variant} = \sum_{I=1}^{N} \text{Time Vari}$$

- Time gap between the generation of two variants of same version i.e., TimeVari
- Average time gap between the generation of two variants of same version i.e.,
- Avg Time Variance = TotalTimeVari/NumVari_ver
- Distribution of changes over various phases of development i.e., PhaseDistribution
- Average distribution of changes over various phases of development i.e.,

$$\text{AvgDistriPhase} = \text{Total No. of changes made over all phases/Total no. of phases}$$

$$\text{Avg Distri Phase} = \sum_{I=1}^{N} \text{PhaseDistribution/N}$$

- Total time to release the version i.e., TotalTimeVer
- Average time taken for release of a version i.e., AvgTimeVer

$$\text{AvgTimeVer} = \text{TotalTimeVer/TotalNumVer}$$

Personnel Metrics

The metrics suite that can be defined at personnel level is as follows:

- Total number of persons involved in version process i.e., TotalPersons
- Total number of changes involved in version process i.e., TotalChanges
- Total number of changes done by every person in absolute, individually i.e., ChangePersons
- Total number of changes done by every person in every phase of development i.e., PersonPhaseTotalChanges
- Average number of changes committed by the persons in a team i.e., AvgChangePerson AvgChangePerson = TotalChanges/TotalPersons

Metrics at Framework Level

The metrics suite that can be defined at framework level is as follows:

- No. of components in a framework i.e., NumCompFrame
- No. of relations in a framework i.e., NumRelaFrame

- Complexity of relations in a framework between various components i.e., CplexRelaFrame
- No. of evolving/evolved frameworks i.e., NumEvoFrame

Metrics at Component Level

The metrics suite that can be defined at component level is as follows:

- Complexity of a component i.e., CplexComp
- Change of complexity over two successive versions i.e., ChangeCplexVer
- Change in internal attributes of a component with respect to

  - Number of modifications done over successive versions i.e., NumModSuccVer
  - Change in size i.e., ChangeSize
  - Change in Cohesiveness i.e., ChangeCohesive
  - Change in Coupling pattern (Complexity of Coupling/Coupling in Complexity Pattern) i.e., ChangeCoupling

Case Study

In order to illustrate the use of the proposed set of metrics, a small component-based system is chosen whose development has been monitored. This system is a library management system composed of seven different components i.e., Person, Member, Student, Teacher, BookIssue, Book and Subject. The system was developed by a team of five persons loosely involved in every developmental activity from establishing the set of requirements component by component till the testing of the component. It has been a deliberate sequential process followed by the developers in order to study not only the behavior of the individual components but also that of the persons involved. It is a controlled set of an experiment specifically designed for study of the use of the proposed set of metrics. The results obtained can be represented with the help of graphs as shown below:

- To find average number of variants to a particular version (Fig. 6)
- To find average time gap between the generation of two variants of same version (Fig. 7)
- To find average distribution of changes over various phases of development (Fig. 8)

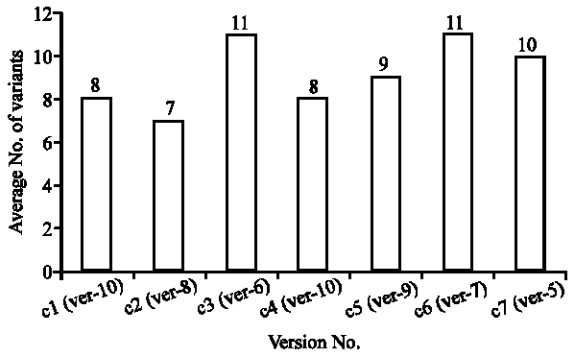

Fig. 6: Average No. of variants to a version

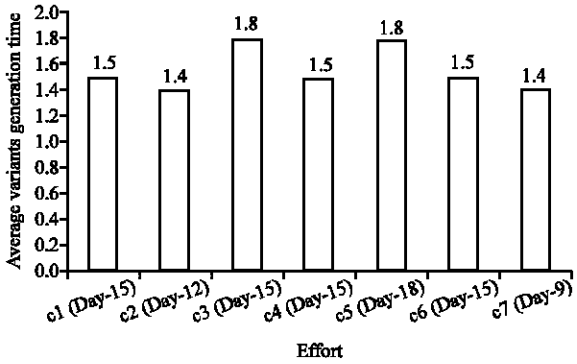

Fig. 7: Average time gap between the generations of two variants of same version

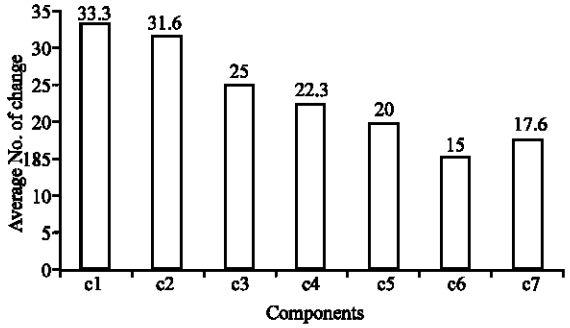

Fig. 8: Average distribution of changes over various phases of development

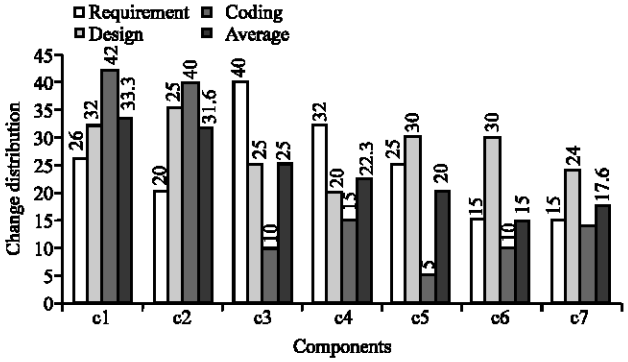

Fig. 9: Distribution of changes over various phases of development

- To find average distribution of changes over various phases of development with respect to component C1 to C7 (Fig. 9)

The number of variants/version and the time/average time spent delivering a variant/version indicates the complexity/length of the job in generation of a version. Greater the number of variants, theoretically, those versions should take greater amount of time to be delivered as is the case with component C3 (Fig. 6, 7). However, components C3 and C5

show a reverse trend which might be due to the complexity and the nature of changes involved. Therefore, the time to release the version is affected by the number, type and the complexity of the changes involved. Average distribution of changes with respect to each component, over various phases of development is shown in Fig. 8.

Another input in the development of the various components with respect to the time to release of the variants comes from the distribution of the errors across various development phases. Component C3 (Fig. 9) took maximum average time to develop because of the greater number of errors encountered in the requirement phase which means it took longer time to get the requirements stabilized. However, the components having spent lesser time in stabilizing the requirements but more time in other phases like Design, Coding, took overall lesser time to be released. This highlights the importance of the early stabilization of requirements. In case of Component C5, the most of the time seems to be spent on the stabilizing Requirements and Design and it actually got developed in surprisingly little coding effort which is according to the trend that stabilized Requirements and Design actually can reduce the coding effort, as is also shown in the case of components C3 and C6 (Fig. 9).

CONCLUSIONS

Component-based systems are becoming increasingly important in software development. The continuous change in these types of systems, demand for an efficient version control mechanism. In this paper, an effort has been made to present a component framework for version management, which is used as a basis of developed version-control tool. A mathematical model is presented which enables the handling of multiple frameworks independently as well as when merged together. More than one framework can share the different variants of same version of the component and any one framework can share the same component with different versions along with their variants.

The properties of the developed version control tool have been discussed and compared with other systems of same type. A supporting set of metrics is also presented to effectively support the working of the proposed component framework and version control tool. However, the developed tool as well as proposed set of metrics needs validation. The next step is to use a wide range of empirical data for validation as well as to undertake a deeper probe into the working of the metrics.

REFERENCES

Coleman, D., P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes, 1994. Object-Oriented Development: The Fusion Method. Prentice-Hall, Englewood Cliffs, New Jersey.

Collins-Sussman, B., B.W. Fitzpatrick and C.M. Pilato, 2006. Version Control with Subversion. For Subversion 1.3, Create Space Publisher, UK.

Cook, S. and J. Daniels, 1994. Designing Object Systems. Prentice-Hall, New Jersey.

Crnkovic, I., M. Larsson and K.K. Lau, 1999. Component configuration management for frameworks. http://www.cs.man.ac.uk/~kung-kiu/pub/wsac99.pdf.

Gergic, J., 2003. Towards a versioning model for component-based software assembly. Proceedings of the international Conference on Software Maintenance, Sept. 22-26, ICSM IEEE Computer Society, Washington, DC., pp: 138-138.

Larsson, M., 2000. Applying configuration techniques to component-based systems. Ph.D. Thesis, Department of Computer Engineering, Malardalen University.

Mauth, R., 1996. A better foundation: Development frameworks let you build an application with reusable objects. BYTE 21(9):40IS 10-13, Sept. 96.

Pieber, A. and J. Spoerk, 2008. A comparative analysis of state-of-the-art component frameworks for the JAVA programming language. Informatikpraktikum I, SS 08. http://cocoon.ifs.tuwien.ac.at/lehre/praktikumsarbeiten/2008_pieber_component_frameworks.pdf.

Thomas, D. and A. Hunt, 2003. Pragmatic Version Control Using CVS. The Pragmatic Programmers, USA., ISBN-10: 0974514004.