Advance Version Control (AVC): A Paradigm Shift from Version Control to Conflict Management

Inderdeep Kaur
Dept. of Computer Science
Guru Nanak Dev University Amritsar

Hardeep Singh
Dept. of Computer Science
Guru Nanak Dev University Amritsar

ABSTRACT
Distance is the major barrier in Global software development. It negatively impacts communication and coordination. Many tools are used to help developers to coordinate their work. One such tool is Software Configuration Management (SCM). But studies have shown that SCM lacks in conflict detection. This weakness has been removed by workspace awareness tool.

This paper describes Advance Version Control tool (AVC), a tool that does versioning of artifacts and actively monitor commit activities of the developer. It is an integration of Configuration Management and Conflict Management. AVC determines conflicts at right time and also provides resolution methods like code sharing. The developed tool allows the developers to focus on their main task i.e. programming without presenting trivial notifications and warnings. AVC also has integrated communication medium to further enhance the coordination.

General Terms
Distributed Software Development, Software Configuration Management, Workspace Management tools.

Keywords
Version control tools, conflict detection, workspace awareness.

1. INTRODUCTION
Globalization of software development has given rise to distributed software development. Distributed Software Development (DSD) or Global Software Development (GSD) refers to style of development in which developers work from geographically dispersed locations. These developers may belong to same organization or may be from different organizations. Also, the dispersed locations may vary from one building to another, from one city to another or from one country to another. In latter case, it is referred as Global software development. The widespread usage of distributed software development is due to its numerous benefits like reduced development cost, reduced time to market, best quality labor and proximity to market [1-6].

Nevertheless, its benefits DSD pose many challenges. These challenges owe to distance. Distance effects three areas of software development communication, coordination and control [2-5]. Due to less information flow in DSD projects coordination requires more effort [3]. Coordination is synthesizing various modules or parts made by different developers into single working unit. When people are working on same area then there are coordination problems.

SCM tools like version control system (VCS) provide impeccable solutions for collaborating teams. It coordinates the activities of individuals or organizations in one form or another for collocated as well as distributed projects [7,8]. It also manages and controls the evolution of software by keeping track of artifacts [9].

Version control systems have undergone a change with shift in centralized to distributed software development [10]. Eric Raymond [11] has categorized the version control systems into three generations ranging from peer-to-peer to distributed approach. Third generation tools are distributed version control systems (DVCS). This category of tools clone full repository on the developer's machine creating private workspace for the developer. Most of the operations including commit can be performed on local machine [10]. Private workspaces help developers to focus on their work without being affected by other developers work [12][13]. Due to this isolation, developers are unaware of each other's work. When the changes from developers' workspaces are reconciled, it could lead to merge conflicts [18]. Microsoft developers reported that in merging operation most of the time is devoted in resolving the conflicts [14]. Brun [32] studied nine open source projects from Github (http://git-scm.com) and concluded that conflicts are frequent, persistent, and appear not only as overlapping textual edits but also as subsequent build and test failures even with modern SCM tools like (Git [20] and Mercurial [21]) which provide enhanced method of automatic merging. Conflicts could be direct (conflicts in the same section of the code) or indirect (conflicts in dependent code also called dependency conflicts). However, if these conflicts are not handled appropriately it can cause merging errors, compilation errors, test suites failure and wrong behavior in shared repository. Also, late discovery of wrong behavior add to complexities and increases cost of correction as well [15]. The rationale behind this is lack of coordination and group awareness [16]. Conflict complexities are inherently less in co-located teams where developers have routine informal communication. But in distributed team's direct communication is rare. In such a setting, some approach is required to enhance the level of awareness to improve coordination [17].

This is place where workspace awareness tools come into existence. Awareness in this context is "an understanding of the activities of others which provides a context for one's own activities" [19]. Being aware of others work enables the developers to detect the conflicts at early stage before they diffuse with the main repository. SCM community has accepted it as challenge and realized the benefits of workspace awareness tools [22-30].Significant contributions have been done by [31, 23, 22, 34, 26] in workspace awareness which assist the developers to detect conflicts early by exchanging information among developers. These tools compliment SCM by coordinating across workspaces still maintaining good isolation.

A new approach, AVC, provides novel solution for conflict management by removing the limitations of currently available workspace awareness tools. The goal of designing AVC is to provide right kind of conflict information at right time by reducing the time wastage of developers in reading redundant notifications and warnings.

2. BACKGROUND

In this section, some of the existing workspace awareness tools are discussed:

2.1 Related Work

Palantir [33]increases workspace awareness by continuously sharing information regarding operations performed by all developers. All the operations are mapped into events and event notification system is used to communicate with developers. Palantir shows which developers are changing which artifacts by how much. Developers are informed of others changes when they switch from one artifact to another. Once modified artifact has been checked-in, Palantir calculates and shows the severity of change (by differencing between old version and new version). Palantir reports the direct conflicts at the level of artifact rather than individual methods or lines of code. For indirect conflicts, it calculates the diff' in class signature and broadcast it. Other developers interpret this to determine whether this change cause conflicts with their local version.

CollabVs[34] extends user interface of visual studio. It provides coordination mechanism to manage conflicts in asynchronous software development. The model constantly looks for dependency conflicts and makes developers aware of these conflicts. Program elements looked for dependency conflicts are classes, methods and files. Developers can choose the granularity of program elements for which dependency conflicts are checked. Also, team members can easily communicate through IM and audio/video session. Code session can be started where developer can browse the remote developers' version to identify the conflicts. CollabVs also provides facility to set watch on remote developer's work, which gives the information when developer finishes its editing task.

Wecode[35] used continuous integration for conflict detection which runs as plug-in of Eclipse. Wecode address direct and indirect conflicts. It constantly merges committed and uncommitted changes of developers. This merged system is analyzed, compiled and tested. Developers' changes are collected at save time. Each developer copy is internally represented as tree. Every file and program element is a node in system tree. Recent changes done to nodes are used for background merging. Nodes which are in conflict are tracked by node change tracking information. Each conflict is reported to the members who changed have the node.

Crystal [36] identifies the conflicts in version controlled artifacts by actually creating a merged artifact. This merged artifact is compiled and tested to determine the conflicts. The resultant conflict information is presented to developer in an unobtrusive manner. Developer can also determine the relationship of his repository with other developers' repository and with master repository. Crystal supports distributed environment and works with mercurial.

Syde[37] is an Eclipse plug-in, which provides conflict awareness using change-centric approach. Syde uses abstract syntax trees (AST) to represent object-oriented system, where nodes of AST are elements of system. Changes are captured at every save operation. Whenever developer changes something in his workspace, syde captures it as an AST operation. Conflicts are detected by comparing AST of developers. This change information is then broadcasted to all the developers. Conflicts are shown in different colours. Red conflicts are considered severe because in this case one entity is already checked-in. Yellow conflicts are moderate in which both the entities are not checked-in.

Cloudstudio[38]is a web-based prototype. It keeps multiple synchronized versions of codebase thus performing functions of SCM. Along with this, Cloudstudio provides real-time awareness system, which makes developers' aware of each other's work before the conflicts occur. This is achieved by editor which indicates other developers' changes by showing changed lines with different colors to the current developer. Developer can switch to see the actual changes. Cloudstudio provides SCM functionality by implementing standard notions like repository, push/pull and branch operations.

2.2 Limitations of current tools

The above mentioned tools have been evaluated on various criteria. During this qualitative evaluation, it has been found that there are several limitations in these tools. This section discusses those shortcomings which have led to design a new tool. All of the existing tools have same goal i.e. to detect the conflicts at early stage and in this process, they produce unnecessary notifications which lead to numerous false positives. An aggressive tool will be producing higher false positives and less false negatives. In an extreme case, a tool detecting conflict at each edit will be producing no false negatives and high false positives. However, most of the tools which are integrated with IDE capture the changes in workspace at the time of edit or saving the file and report the conflict. Thus, these tools produce higher false positives because there is high probability that conflict disappears till the final changes done before commit [39].

Moreover, plethora of notifications like who is changing which file or which element of the file, overload the developers and distract them from doing actual work and hence affect the productivity [42, 43].

Another downside of these tools is that developers feel privacy invasion when each change is monitored and viewed by others. They want only final commits to be viewed by others. Also, conflict discovery requires conflict computation and communication cost. So, conflict discovery at each edit and save, involves lot of cost [40].

3. PROPOSED FRAMEWORK

From current state of art, it is found that still there is a need for a tool which will discover the conflicts at right time. Keeping this in mind, a novel approach for version control has been proposed which has conflict detection and conflict resolution capability also. Version Control System and Workspace Awareness have been combined because SCM is the best place to monitor the activities of developers where they are collaborating. This tool is named as Advance Version Control (AVC). It has been designed to manage the evolution of the software and to detect the conflicts at right time with reduced false positives and unnecessary notifications.

3.1 AVC Framework and model

Tool has been designed to deal with five aspects of collaborative software development:

- Software versioning/version control
- Conflict detection

- Communication among developers.
- Code sharing.
- Group Awareness.

Software versioning is version control module. AVC is distributed version control system. AVC version control module has been adapted from Git [20]. After evaluating and comparing various distributed version control tools, it has been found that Git has been designed with excellent features, so it was natural to adapt it [41]. It allows the developers to create new repositories or clone existing repositories created by others. Cloning allows developers to create a private workspace on their system. Developers can make changes to files in their workspace, AVC track the changed files and *stage* them. Staging is one step prior to commit which assures that changed files need to be committed. *Commit* operation assigns new Revision ID to changed files. Pull feature is also there which allows developers working in distributed environment to see each other's work. New version of project can be created by integrating work from different developers.

Conflict Detection module is responsible for detecting the conflicts and presenting it to developer. AVC allows conflicts to be detected before commit but after the changes are finalized. Detecting the conflicts after the changes are confirmed saves the developers from trivial notifications and warnings. When a file is changed and staged for commit, a potential conflict is shown if the same file is staged or committed by some other developer. AVC will show the difference of your version of file and conflicting file. If there is no conflict it can be ignored. If there is actually a conflict then any of the conflicting developer rollback his/her code or they can start code sharing session to mutually resolve the conflict.

Sometimes it is difficult for a developer to resolve conflict without communicating with other developer, especially when both the developers are having conflict in same lines of code. Communication in AVC is carried out using messaging. It can also be used for routine communication to understand the code like who is changing which part of the code and also it can be used to send message for starting the code session.

AVC allows code sharing session to be opened independently or when conflict arises. The motive of adding this module is conflict recovery by using synchronous editing session between two developers.

Group Awareness works in two ways. For Project owner, it provides information about how many contributions are done by each contributor. For Project contributor, it provides information about which contributor has maximum number of commits in each file. This information helps developer to know which contributor has more expertise in the code. Fig 1 shows the working of AVC model.

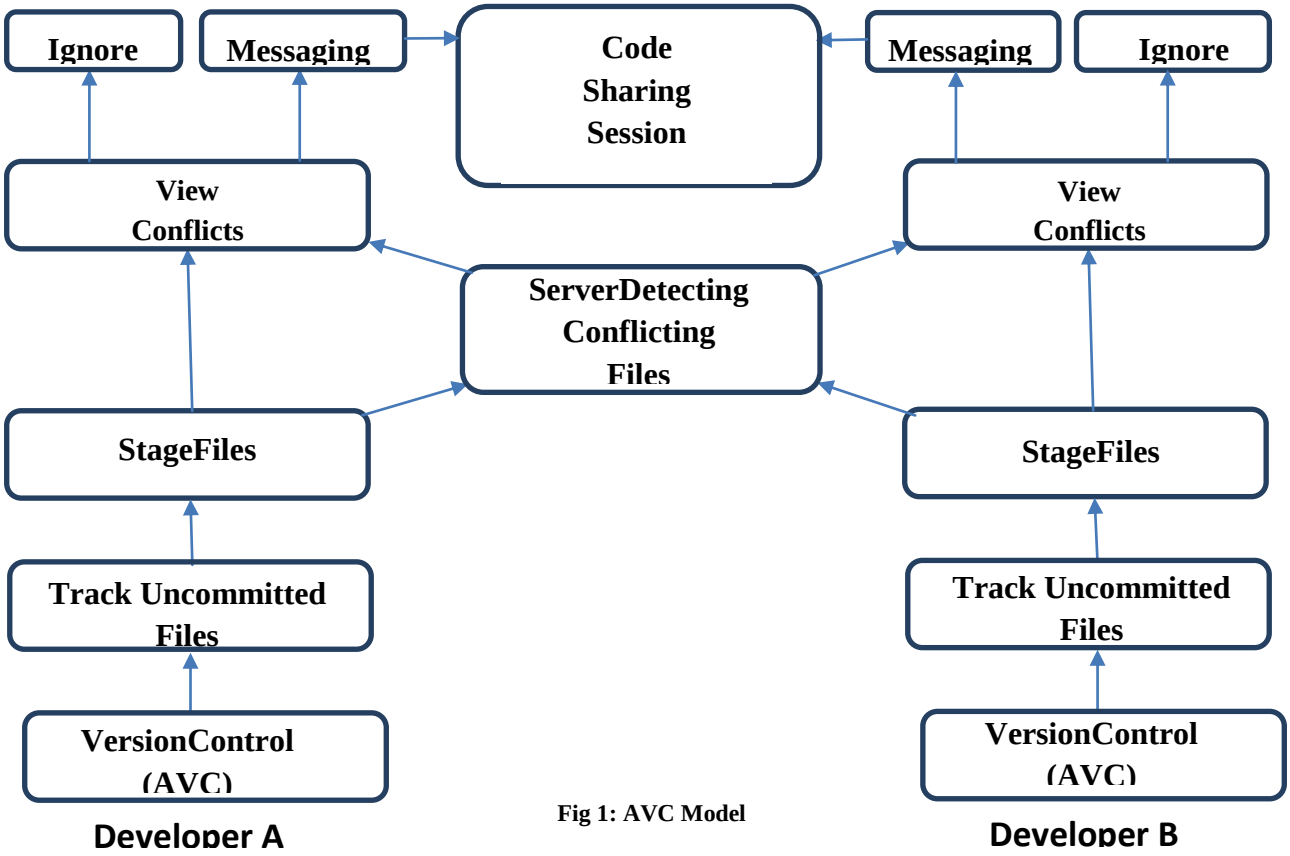

Fig 1: AVC Model

3.2 Features of AVC

AVC is client-server application. It is entirely written in Java in NetBeans. It uses JMerge utility for viewing the difference and merging of files.

- New AVC repository can be created or existing repository can be cloned.
- Track the changed but uncommitted files and *stage* them.
- Commit the file and creates new revision of the file in the repository.

- Creates the change log in the file with author userid, date and time of change and comments for change.
- Branches can be created in local repository.
- Other developers work can be pulled, either single file or whole repository.
- Recent commits of any developer can be viewed of any selected project.
- Diff n Merge(Auto) shows all the differencing files in two selected projects.
- Diff n Merge(Manual) shows file by file difference.
- AVC also shows the difference between subsequent versions of a file.
- New version of project can be created by integrating different developers work.
- Messaging service can be used for project discussions and initiating Code session.
- Conflicts are detected real time.
- AVC detects potential but perpetual conflicts.
- Code sharing session can be used in conflict recovery. Developers can also open this session when they are closely working on the same code.
- Group awareness system of AVC gives information about number of commits in each file and commits done by each contributor.

3.3 Working of AVC

AVC is an optimistic version control system which allows for parallel development. Developer has to create Username and password for Login. After successful login, AVC will open main window as shown in figure 2. This main window will provide all the options useful for developers for versioning and conflict management.

3.3.1 Repository creation and cloning

Option (1) *Create Repository* is used to create a new repository by developer. AVC will convert normal project on developer's system to AVC repository on server. Developers can contribute to other developer's project by cloning his/her repository from server to their local machine. This can be done by using option (2) *Search Repository*. Projects can be searched by using some keywords. Once the project is searched, it can be cloned.

3.3.2 Staging and committing the changed files

Once the developer change files, they need to be committed. This commit is done by AVC in local repository as well as silently on server also. But AVC adds one step prior to commit i.e. staging. When developer changes files, AVC track those changed files and stage them. Staging confirms that these files are going to be committed. This can be done using *Staging* option (3) in figure 2. This option will open the window as shown in figure 3. Changed files are tracked and staged by *Tack Uncommitted Files* option (1) in figure 3. Developer performs the commit on staged files by giving comments for commit. A new version is assigned to the committed file. The information about commits and the committers is saved as log in changed file. This log can be viewed by any developer.

3.3.3 Information about contributors

Owner of the project can any time view the contributors of the project. This information can be obtained by using *MyContributors* option (4) in figure 2. Developer can also view all the projects for which he/she is contributor. This can be done from option (5) *MyContribution* in figure 2.

3.3.4 Creating Branches

AVC allows developers to create branches in their private repositories by making parallel line of development from main repository and test the changes separately. Any number of branches can be created in AVC. Branch is created at the time of commit.

3.3.5 Creating New Version

Most of the time owner of the project is also the integrator. AVC allows the owner to get commits of all the contributors and create a new version of the project and upload it. *ViewCommits* (6) in figure 2 is used by Project Owner/Integrator to download the commits of all the contributors, merge them as required, produce a new version and upload it. A new version can be uploaded using *Upload New Version* option (7) in figure 2.

3.3.6 Pull Files between developers

While using distributed version control system, developers have private workspaces. So to see other developer's work, they need to pull it. AVC provides option that any contributor can Pull work of other contributor with his/her permission. This can be done using *Pull* option (8). Developers can pull a single file or a whole project.

3.3.7 Viewing Difference and Merging of files

In AVC JMerge tool has been used for merging of java files. Two types of merging facility have been provided. *Diff n Merge Auto* option (9) and *Diff n Merge manual* option (10). First one compares the two folders and automatically detects conflicting files and shows the conflicts in diffviewer. After removing the conflicts, files can be merged manually. The latter option is used to manually select the files to view the difference or merge them.

3.3.8 Conflict Detection

AVC conflict detection for direct conflicts works at the time of staging. When a developer changes the file/files, AVC tracks the changed files and stage them. At that time, AVC will check if the same file/files are staged or committed by other contributor, then they are shown as conflict. Conflicting developers for the staged file are shown in figure 4. AVC allows you to view the difference of your files and conflicting files. If it is not a conflict, it can be ignored by developer and he/she can continue with the commit. If there is actually a conflict, developer can modify his code or can start code sharing with conflicting developer to mutually resolve the conflict. Code sharing session is shown in figure 5.

3.3.9 Indirect conflicts with compiler

AVC detects indirect conflicts with compilation. Developer can integrate other contributor's work with their own work in different directory and compile the changes to determine the dependency conflicts. *Get Recent Commits* option (11) is used to view the commits of the other developers. Commits can also be downloaded, which can be integrated with developers own work in separate directory using *Diff n Merge(manual)*and can be compiled using *compiler* option (12). Resulting compilation errors depict dependency conflicts.

Compiler can also be used to test developers own changes with the rest of his/her repository before commit.

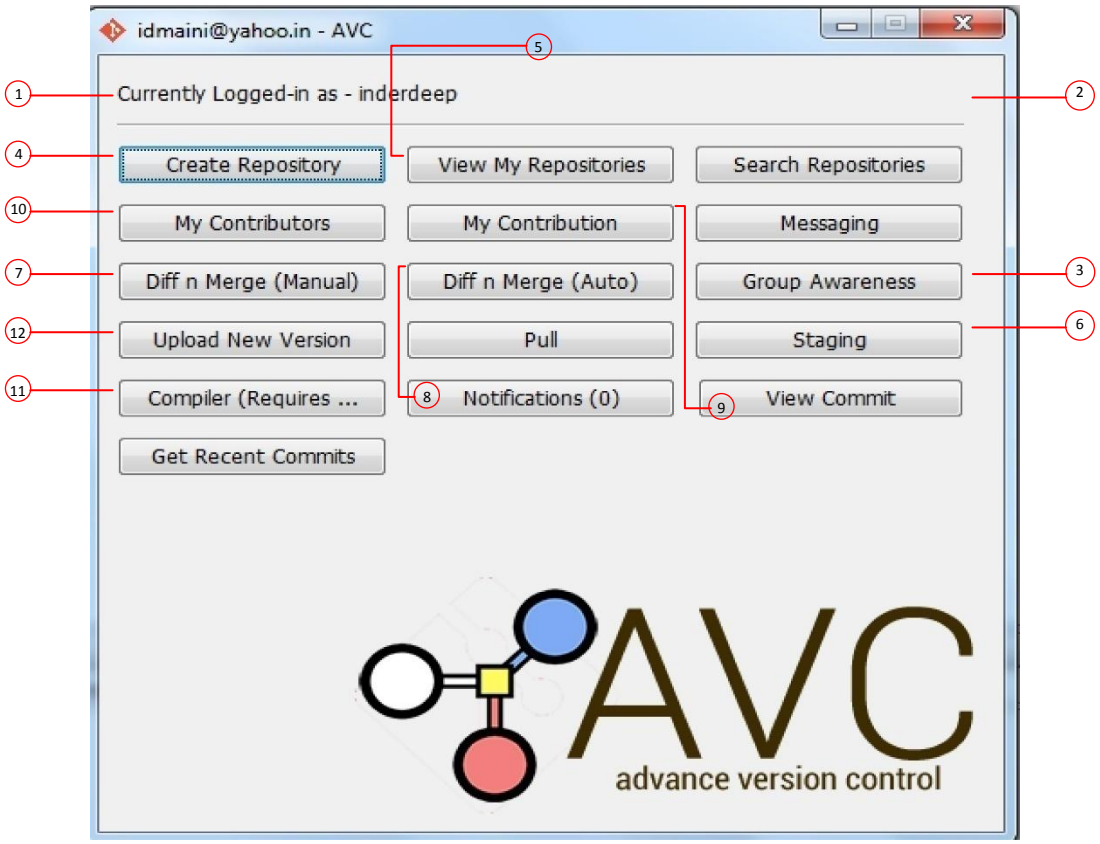

Fig 2: Main window of AVC

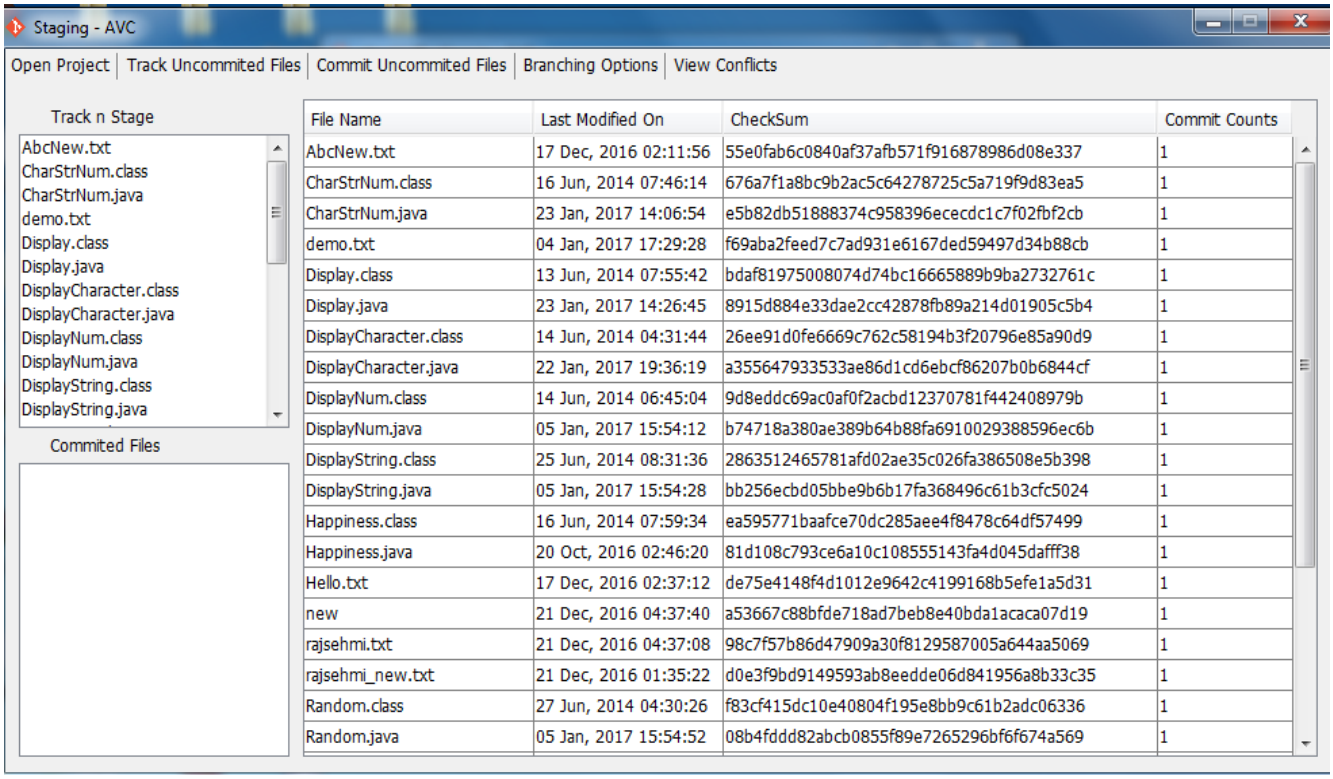

Fig 3: Stage, track and commit window of AVC

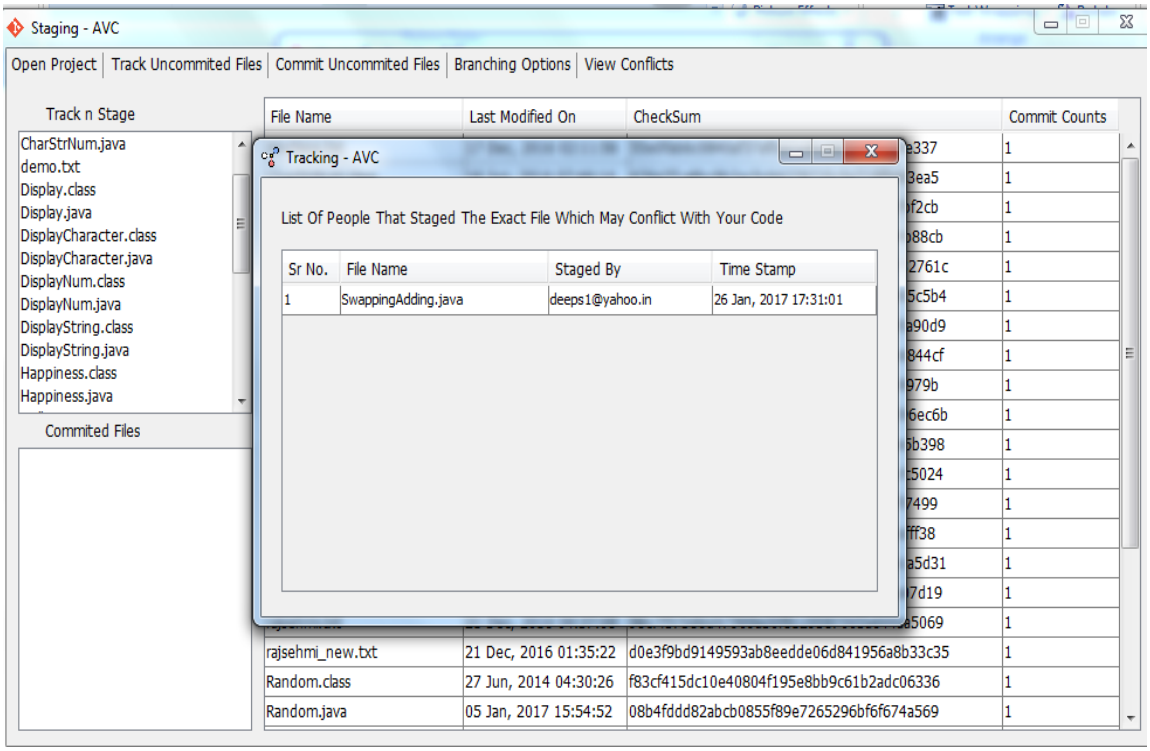

Fig 4: Conflicts detected at the time of staging

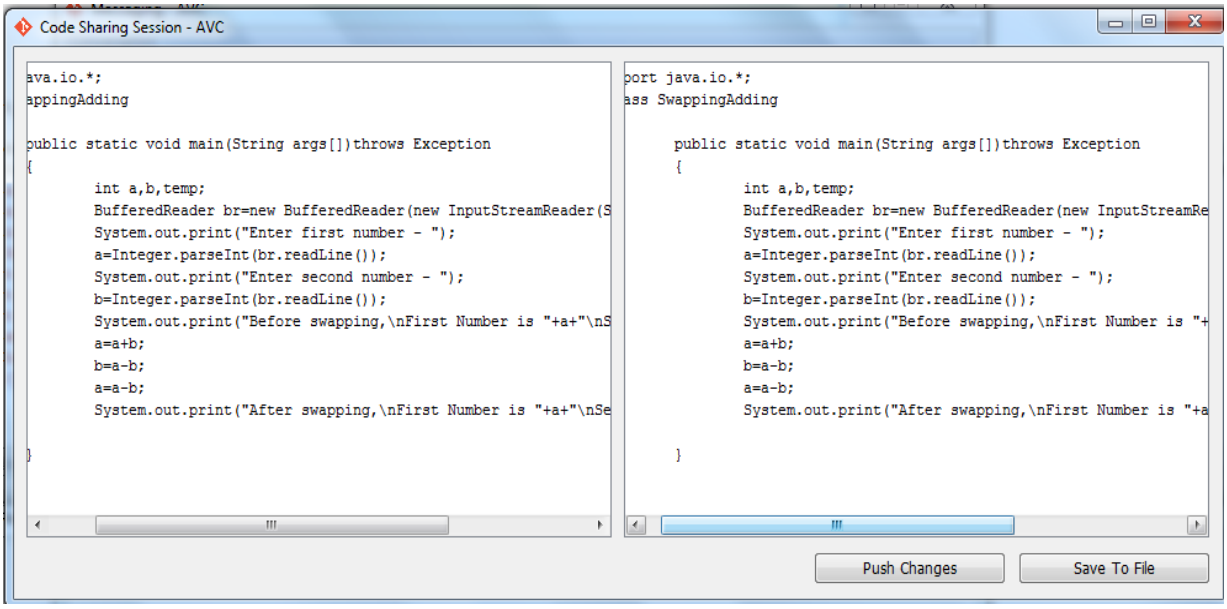

Fig 5: Code sharing between two developers

4. COMPARISON OF AVC WITH OTHER WORKSPACE AWARENESS TOOLS

This section compares AVC with other workspace awareness tools. Comparison is shown in table.1. Various features on the basis of which comparison has been done are given below:

1. **Setting watch in other workspace:** Developer can set a watch in other developer's workspace to know when some work under progress is completed.
2. **Synchronous Editing**: It allows the developers to open local and remote version of the code side-by-side. Changes done at one end are visible to other end on real time. This facility allows the developers to mutually resolve the conflicts.
3. **Importing remote code:** Developerscan import or pull the conceivable conflicting changes of other developer, integrate them with local changes and test them.

4. **Setting watch in other workspace:** Developer can set a watch in other developer's workspace to know when some work under progress is completed.
5. **Dependency Conflicts:** This parameter describes up to what granularity dependency conflicts are determined. Conflicts are of two types that are direct and indirect conflicts. Direct conflicts are also called textual conflicts, occurs when work of two developers who have changed same piece of code in different ways is merged.

Indirect conflicts are also called dependency conflicts.Wheneverthere is a changein some API or method, code using it get affected and can cause conflicts. One obvious method to determine indirect conflicts is by compilation which discovers all dependency conflicts. Some tools may use other methods like dependency graphs to determine dependency conflicts.

6. **Works with Version Control**: This parameter defines either tool is working independently or interacting with some version control.
7. **Testing on merged code:** This parameter determines whether the tool is performing testing on integrated code to discover more conflicts.
8. **Integration with SCM or IDE:** This parameter determines whether workspace awareness is integrated with Configuration management, Integrated development environment or working independently.
9. **Conflict discovery before or after check-in:** This parameter determines whether conflicts discovery is done before the commit performed by developer or after the commit.
10. **Code sharing:** This is a medium used for communication between developers via code. By using code sharing session, developers can resolve the conflict by mutually agreeing on code. This parameter determines whether tool is using code sharing facility to resolve the conflicts.
11. **Conflict information to conflicting developers:** Workspace awareness tools collect the developer's workspace information and share it with other developers. When a conflict is detected, it is communicated to developers. This parameter determines whether conflict information is communicated to conflicting developers or broadcasted to all the developers of the team.
12. **Time of conflicts discovery:** Workspace awareness tools are designed to collect the conflict information at some particular developer action like at the edit time, save action or at the time of commit.

5. CONCLUSIONS

The purpose of using version control tool is to integrate and coordinate the work of different developers located at geographically dispersed locations. But these tools have weakness in handling the conflicts and these unhandled conflicts are reported at delayed stage. Workspace awareness tools have been designed by researchers to compliment SCM by providing the conflict information at early stage. However, these tools overload the developers with notifications, which makes difficult for developer to find real conflict information. The purposed tool, AVC, integrates SCM and workspace awareness. AVC conflict management detects the potential conflicts at right time without creating disturbance to the developer. It has easy to use versioning system and provides full control to developer for communication. The group awareness module of AVC provides enhanced assistance in development work.

Another contribution done by this paper is analyzing and comparing the existing workspace awareness tools, which helps the team to select the particular tool according to project requirement.

This research work can be improved in future by adding automatic merging techniques instead of manual merging. At present, AVC is detecting higher order conflicts by compilation. This can also be automated by using dependency graphs or some other techniques.

6. ACKNOWLEDGMENTS

Miss kaur is thankful to Department of Computer Science, Guru Nanak Dev University, Amritsar, providing me the opportunity to carry out my research work.

Table 1. Comparison table

Sr.no	Features	CollabVs	Crystal	Palantir	Syde	Wecode	Cloudstudio	AVC
1.	**Synchronous Editing**	Yes	No	No	No	No	Yes	Yes
2.	**Importing remote code**	Yes	No	No, shows diff only	No, only the difference of two developers work with conflict is shown.	No	Yes	yes
3.	**Setting watch on others work.**	Yes	No	No,continuous event generation.	No	No	NA	No
4.	**Dependency conflicts**	File, method, class or interface and method	Public class variables and methods.	Class signature	Yes(with compilation)	Yes(with compilation)	Yes(with compilation)	Yes(with compilation

5.	**Works with Version Control**	No	Yes, Mercurial	CVS and Subversion.	CVS and SVN	With mainstream VCS.	Itself	Itself
6.	**Testing on merged code.**	No	Yes	No	No	Yes	Yes	No
7.	**Integration with SCM or IDE**	Integrated with Visual Studio	Independent tool	Plug-in of Eclipse	Integrated with eclipse.	Eclipse plug-in	Itself	Independent version control
8.	**Conflict discovery before or after check-in.**	Before Check-in	Before and after	Before check-in	Before and after both	Before	Real-time	Before and after check-in
9.	**Code sharing**	Yes	No	No	No	No	Yes	Yes
10.	**Conflict information to conflicting developers**	Only conflicting members	Repository relationship shared with all	Broadcast	Broadcast	Only affected members	Only conflicting members.	Only conflicting members
11.	**Time of conflicts discovery**	Editing time	Edit time	Save action	Save action	Save action	Edit time	staging

7. REFERENCES

[1] Carmel, E. 1999. Global software teams: Collaborating across borders and time zones. Prentice Hall upper saddle river, NJ,USA.

[2] Lanubile, F. 2009. Collaboration in distributed software development. Software Engineering, Springer, 174-193.

[3] Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., Conchúir, E. O. 2005. A Framework for considering opportunities and threats in distributed software development. In Proceedings of the International Workshop on Distributed Software Development, Austrian Computer Society.

[4] Gumm, D. C. 2005. Distribution Dimensions in Software Development Projects: A Taxonomy. IEEE Software 23(5), 45–51.

[5] Ågerfalk, P. J., Fitzgerald, B., Holmström, H. andConchúir, E. O. 2008. Benefits of Global Software Development: The Known and Unknown.In Proceedings of ICSP'08the Software process, international conference on Making globally distributed software development a success story. 1-9, Springer-Verlag.

[6] Herbsleb,J.D.,Moitra, D. 2001. Global Software Development. IEEE Software, volume 18, Issue 2, 16–20.

[7] Software configuration management: A practical guide.https://energy.gov/sites/prod/files/cioprod/documents/ scmguide.pdf.

[8] Bendix, L., Magnusson, J.,Pendleton, C. 2012. Configuration management stories from distributed software development trenches. IEEE Seventh International Conference on Global Software Engineering (ICGSE).

[9] Grinter,Rebecca.E.1996. Supporting Articulation Work Using Software Configuration Management Systems. Computer Supported Cooperative Work,Volume 5, Issue 4,447- 465.

[10] Clatworthy, I. 2007. Distributed version control: Why and how. In Proceedings of Open Source Development Conference (OSDC).

[11] Raymond, E. Understanding version-control systems. Retrieved from http://www.catb.org/esr/version-control/version-control.htm.

[12] De Souza,Cleidson, R.B., Redmiles, D. and Dourish, P.2003." Breaking the code", moving between private and public work in collaborative software development. In Proceedings of ACM SIGGROUPInternational conference on supporting group work GROUP'03, Florida, USA, 105-114.

[13] Conradi,R. and Westfechtel,B. 1998.Version Models for Software Configuration Management. ACM Computing Surveys, vol.30,232-282.

[14] Bird, C. and Zimmermann, T. 2012. Assessing the value of branches with what-if analysis. In proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, North Carolina.

[15] Levin,S. and Yehudai, A. 2015.Alleviating Merge Conflicts with Fine-grained Visual Awareness.arXiv preprint arXiv:1508.01872.

[16] Fauzi,S.S.M., Bannerman, P.L. and Staples, M. 2010. Software Configuration Management in Global Software Development: A Systematic Map. In Proceedings of 17th Asia Pacific Software Engineering Conference.

[17] Chen, C. Zang, K. 2013. Team Radar: A Radar Metaphor for workspace Awareness. Evaluation of Novel Approaches to Software Engineering. ENASE 2011. Communications in Computer and Information Science,vol 275. Springer, Berlin, Heidelberg.

[18] Sarma, A., Noroozi, Z. and Hoek Andre, V, D. 2003. Palantír: raising awareness among configuration management workspaces. In Proceedings of the 25th International Conference on Software Engineering, May 03-10, Portland, Oregon.

[19] Dourish, P. and Bellotti, V. 1992. Awareness and Coordination in shared Workspaces. In Proceedings of ACM Conference on Computer-Supported Cooperative Work,Tronto,Ontario,Canada.

[20] "GIT, a free, open source, distributed version control system," http://git-scm.com/, [Online; accessed 11-April-2011].

[21] "Mercurial is a free, distributed source control management tool."http://mercurial.selenic.com/, [Online; accessed 02-April-2011].

[22] Sarma, A. and Hoek, A. vander. 2002. Palantir: coordinating distributedworkspaces. In Proceedings of 26th International Computer Software and Applications(CMPSAC.2002), pp. 1093 – 1097.

[23] Jacob, T.Biehl., Mary, C.,Greg, S. and George, G. R. 2007. FASTDash: A visual dashboard for fostering awareness in software teams. In proceedings of SIGCHI conference on Human factors in Computing systems ACM, pp 1313-1322, San Jose, USA.

[24] Lucia, A.D., Fasano, F., Oliveto, R. and Tortora, G. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. ACM Transactions on Software Engineering and Methodology, volume 16, issue 4.

[25] Bang, J.Y., Popescu, D., Edwards, G., Medvidovic, N., Kulkarni, N., Rama, G.M., and Padmanabhuni,S. 2010. CoDesign: a highlyextensible collaborative software modeling framework. In Proceedings of ACM/IEEE 32nd International Conference on Software Engineering, vol. 2, pp. 243 –246

[26] Hattori, L. and Lanza, M. 2010. Syde: a tool for collaborative software development. ACM/IEEE 32nd International Conference on Software Engineering,vol. 2, pp. 235 –238.

[27] Lucia, A.D., Fasano, F., Oliveto, R., and Tortora, G. 2010. Fine-grained management of software artefacts: the ADAMS system. Software:Practice and Experience, vol. 40, no. 11, pp. 1007–1034.

[28] Dam, H. K and Ghose, A. 2011. An agent-based framework for distributed collaborative model evolution. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, NY, USA, pp. 121–130.

[29] Huyen, P.T.T. and Ochimizu, K. 2012. A Change Support Model for Distributed Collaborative Work. CoRR,

[30] Costa, C., Mutra, L. 2013. Version Control in Distributed Software Development*:* A Systematic Mapping Study. In Proceedingsof ICGSE IEEE 8th International Conference on Global Software Engineering, page 90-99.

[31] Guimaraes, M.L. and Silva, A.R. 2012. Improving early detection of software merge conflicts. In Proceedings of International Conference on Software Engineering ICSE'12, pages 342–352. IEEE Press.

[32] Brun, Y., Holmes, R., Ernst, M. D. and Notkin, D. 2011. Proactive detection of collaboration conflicts. In Proceedings of the 8th Joint Meeting ofthe European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE11), pp.168–178.

[33] Sarma, A. and Hoek, A. vander 2002. Palantir: coordinating distributedworkspaces. In Proceedings of 26th International Computer Software and applications (CMPSAC)pp. 1093 – 1097.

[34] Dewan, P. and Hegde, R. 2007. Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. In Proceedings of the 10th European Conference onComputer Supported Cooperative Work (ECSCW '07), pages 159–178. Springer, London.

[35] Guimaraes, M.L. and Silva, A.R. 2012. Improving early detection of software merge conflicts. In Proceedings of International Conference on Software Engineering ICSE'12, pages 342–352. IEEE Press.

[36] Y, Brun., R, Holmes., Ernst, M. D., and Notkin, D. 2011. Crystal: Precise and Unobtrusive Conflict Warnings. In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference Foundations of Software Engineering (ESEC/FSE'11), pp.444-447, NY, USA.

[37] Lanza, M., Hattori, L., and Guzzi, A. 2010. Supporting collaboration awareness with real-time visualization of development activity. In Proceedings of 14th European Conference on Software Maintenance and Reengineering (CSMR), 202 –211.

[38] Nordio, M., Estler, H.-C., Furia, C. A. and Meyer, B. 2011. Collaborative software development on the web. arXiv:1105.0768v3.

[39] Hattori, L., Lanza1,M., D'Ambros, M. 2012. A qualitative user study on preemptive conflict detection. In Proceedings of ICGSE,IEEE Seventh international conference on Global Software Engineering.

[40] Dewan, P. 2008. Dimensions of tools for detecting software conflicts. In Proceedings of ACM international workshop on Recommendation systems for software engineering (RSSE'08). Pp 21-25, NY, USA.

[41] Kaur, I., Kaur, P. and Singh, H. 2017. A Comparative Study of Distributed Version Control Tools. In Proceedings of 5th International conference on Advancements in Engineering and Technology (ICAET-2017), ISBN No. 978-81-924893-2-2.

[42] Damian, D., Izquierdo, L. et al. 2007. Awareness in the Wild: Why Communication Breakdowns Occur. In ICGSE '07: Inter.Conf. on Global Softw. Eng. IEEE Computer Society,81–90.

[43] Kim, M. 2011. An Exploratory Study of Awareness Interests about Software Modifications. In CHASE '11: Workshop on Cooperative and Human Aspects of Softw. Eng. ACM, 80–83.