

# The Use of Information Visualization to Support Software Configuration Management\*

Roberto Therón<sup>1</sup>, Antonio González<sup>1</sup>, Francisco J. García<sup>1</sup>,  
Pablo Santos<sup>2</sup>

<sup>1</sup> Departamento de Informática y Automática, Universidad de Salamanca,  
Plaza de la Merced s/n. 37008, Salamanca, Spain  
{theron, agtorres, fgarcia}@usal.es

<sup>2</sup> Códice Software, Edificio Centro, Parque Tecnológico de Boecillo  
47151, Valladolid, Spain  
psantosl@codicesoftware.com

**Abstract.** This paper addresses the visualization of the collaboration history in the development of software items using a simple interactive representation called Revision Tree. The visualization presents detailed information on a single software item with the intention of supporting the awareness of the project managers and developers about the item evolution and the collaboration taking place on its development. We considered that repositories of Software Configuration Management tools are the best information source to extract relevant information dealing with the relationships between the programmers and software items, as well as information regarding the creation of baselines, branches and revisions, and useful date and time details for the arrangement of the development timeline and collaboration representation.

**Keywords:** Software Configuration Management (SCM), Information visualization, Focus + context, Time line, Polyfocal display, Interaction, Revision Tree

## 1 Introduction

The software development process and the collaboration that it involves are difficult to understand and represent, due to the large number of software items that constitute a software product. Moreover, the collaboration taking place in the development of each item is concurrent and may be distributed across several geographical locations. Software Configuration Management (SCM) controls the evolution of complex systems [1] taking into consideration the communication at every level of the organization as well as the changes of code and documentation. To accomplish this purpose, the tools supporting such a process must provide services for the management of the component database, enhancing the environment of the developers, managing concurrency and collaboration, and recording changes

---

\* This work was supported by the Education and Science Ministry of Spain under projects TSI2005-00960 and TIN2006-06313.

including time, date, which modules were affected, how long the modification took and information about who did the change.

The IEEE Standard 828-1990 [2] states that “SCM activities include the identification and establishment of baselines; the review, approval, and control of changes; the tracking and reporting of such changes; the audits and reviews of the evolving software product; and the control of interface documentation and project supplier SCM”. Hence, the importance of SCM repositories as the information source to extract the collaboration activities taking place during project developments, as well as key information as the identification and establishment of baselines and revisions and the tracking of changes including dates and times. However, in spite of the richness of this data source and decades after the first SCM systems were released, there is an important lack of mechanisms with which to convey, by means of proper representations, how the contribution and collaboration of team members occurs in a particular project.

In recent years, the field of information visualization has played an important role in providing insight through visual representations combined with interaction techniques that take advantage of the human eye’s broad bandwidth pathway to the mind, allowing experts to see, explore, and understand large amounts of information at once [3]. Traditionally, the software development process has been a subject of interest for information visualization practitioners. Thus, the software visualization community is providing excellent results which are being featured in main stream IDEs. Nevertheless SCM tools can still be enhanced by using highly interactive visualizations rather than mere “static” representations.

The interactive visual solution we propose in this paper considers both space and time strategies: the space strategy uses layout and graphic design to pack appropriate information in one view, while the time strategy uses view transitions to spread information over multiple views [4]. Additionally, we take into consideration several techniques to support navigation, interpretation of visual elements and understanding relationships among items in their full context [5].

There are also many information visualization techniques, each one with its advantages and disadvantages; the use of a sort of combination to provide a real solution to end users is very frequently required. Spence [6] and Card et al. [7] provide excellent surveys of information visualization mechanisms and techniques. We support our visualization through the use of a grid-based structure, selection, navigation, filtering and zoom interaction mechanisms, in addition to polyfocal display, a tree hierarchy (a directed graph) and a time line as visualization techniques.

We considered what is going on in the project in our design, who else is working on the project, what they are doing, how long they have been working on a revision, how their work may impact the work of others and the overall framework designed by Storey in [8] for describing the visualizations of human activities in software engineering.

This paper is devoted to present the first contribution to the SCM tool (PlasticSCM) developed by Códice Software (<http://www.codicesoftware.com>); an interactive 2D visualization, named Revision Tree, which allows visualizing the contributions of the team members, through several revisions, baselines and long periods of time, on the same item or document within the software project. This way, the rest of the paper is organized as follows: Section 2 reviews some related works

applied to the visualization of software evolution and software visualization techniques; Section 3 discusses the design of the Revision Tree visualization; Section 4 analyzes a case study in which the results of the Revision Tree are compared with the ones offered by a 3D version tree present in the current version of PlasticSCM tool; and, finally, Section 5 discusses the conclusions and future work.

## 2 Related work

Considerable work has been dedicated to study the software visualization and information visualization areas. Gracanin [9] states that Software Visualization is “a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration”. As a consequence, it is important to identify the tasks that will be performed by the visualization as well as its scope and content, who the audience will be, what data source is going to be represented, how it will be represented, which medium will be used for the representation, the forms and techniques that will be used by the presentation and how the user is going to interact with the visualization.

Although we concentrate on the evolution of individual items and the collaboration of software teams on its development, in this section we will review some useful ideas that have been applied in the visual representation of Software Configuration Management tools repositories.

Xie et al. [10] list a set of questions that can be used to guide the design of visualizations of SCM tools repositories; for the purposes of this paper, it is relevant to determine which authors worked on the same file, when a modification was made and how many authors worked on the release of the system.

Moreover, Eick et al. [11] accurately state that a fundamental problem in visualizing software changes is to choose effective visual representations or metaphors and review some of them, as well as some combinations showing different data perspectives filtered by developer, basic statistics about changes, size of the changes, activity carry out by developer, etc.

Voinea and Telea [12] support the idea that software configuration management repositories are valuable for project accounting, development audits and understanding the evolution of software projects. We strongly agree with these authors about the richness of software repositories; thus, the effective design of the repository of SCM tools can provide information about the development process that is not possible to acquire from any other source and through a well-designed visualization, it is possible to navigate the repository data and get an insight of what is going on in the project. The same authors also propose two visualizations for software management configuration repositories in [13] and [14]. Those proposals demonstrated that the adequate use of 2D visualizations in conjunction with colors and textures contribute to the development of powerful multidimensional visualization solutions.

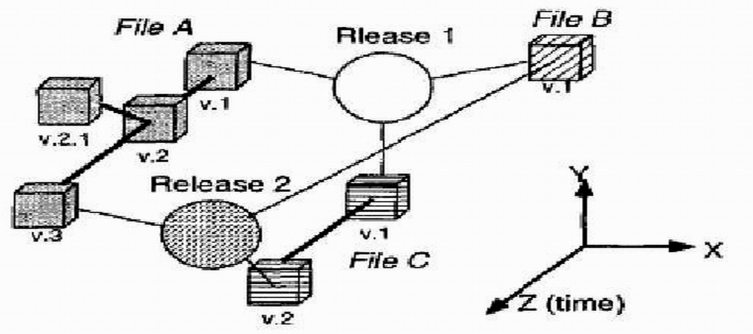
Gall et. al. [15] developed an interesting approach using 3D representations and color coding applied to software evolution through the time, thinking over structure and attribute changes. The attributes are the revision number, item size and

complexity. This approach visualizes the version and each item attributes every time, using one color for each attribute.

There are several proposals addressing the representation of temporal spaces using many different structures. Morris et al. [16] worked with the visualization of temporal hierarchies plotting research documents along a horizontal track in the time line and placing related documents according to the hierarchical structure produced by the clustering phase. Card et al. [17] developed a visualization that allows exploring hierarchies that change with time by using searches, navigating through a hierarchical presentation and filtering results with the assistance of a time slider control. Therón [3] proposed a tree-ring metaphor to represent hierarchical time-based structures and applied it to browse and discover relationships in the history of computer languages. Kumar and Garland [18] proposed a solution for the visualization of time-varying graphs, where the users can slide to different time periods to explore the graph or discover trends interacting with the presentation.

The visualization presented by Lanza in [19] deals with the visualization of software attributes throughout the time using an evolution matrix with variable rectangular-sized boxes inside each cell; the width of the boxes represents the number of methods and the height the number of attributes in the class. This visualization method is powerful and could be improved borrowing some ideas about colors and textures from [15].

At this point, it is relevant to reference the work developed by Koike [20][21], which describes a representation, called VRCS, that shows the evolution of items from the repository of the software management configuration tool. On this visualization, each software item is represented by using two dimensions and the overall visualization with three dimensions, as illustrated in figure 1. This visualization will be analyzed further, in the presentation of the case study.



**Fig. 1.** Software history visualization using 3D presentation for several items

Finally, Perforce (<http://www.perforce.com/>) is a software configuration management system that includes a visualization module; a sample of which is displayed in figure 2. The visualization offered by this tool is two dimensional and uses a graph to show the relationships between baselines, branches and revisions. It features an overview + detail approach rather than a more convenient focus + context approach [22] and will also be discussed in section 4.

### 3 Revision Trees: visualization of the collaboration history of software items

In this section we propose a 2D representation for the collaboration history of software items. The Revision Tree was designed to visualize the contributions of the team members through several revisions, baselines and long periods of time, on the same item or document within the software project. In this context it is important to consider that the evolution of every software item implicitly holds a temporal attribute, which is the most important and critical element needed to understand the software development process of any system. The problem at hand presented several challenges that were addressed in the proposed visualization: the representation of large revision trees, where the baselines have several branches and each branch many

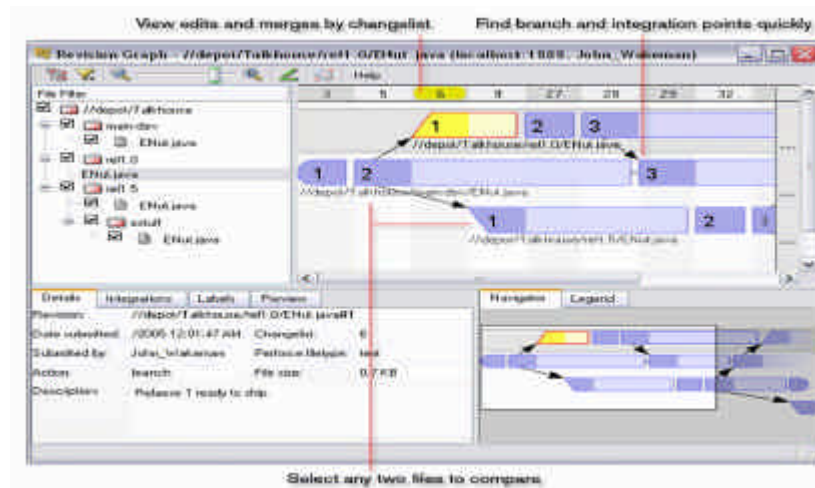


Fig. 2. Visualization of the evolution of a software item with Perforce.

item revisions; the navigation through the version tree offering a focus + context view; support to interactivity to enable the inspection of more than one baseline at a time, exhibit the collaboration of developers for every baseline and correlate all the information with the time line. The full evolution is displayed for a complete analysis in figure 3 (it was turn around due to page constrains), and a piece of this representation is examined further in figure 4.

At this point, it is important to highlight that we decided to use a grid-based structure because it provides an intuitive mechanism to visualize the working relationship between authors and baselines by using the rows to represent the authors and the columns for the baselines (when changes expand during a number of baselines, the column is named after that interval of baselines). Moreover, grid and matrix structures are widely known by developers and the cells can be used as

containers for the drawing of nodes of the directed graph representing the flow of revisions for the item.

Figure 3 depicts three sketches of the same revision tree and figure 4 shows a zoom through a piece of this design in order to better review the details. The first sketch (the one on the top) exposes the normal vista of the design; it uses variable width columns to accommodate the revisions in each baseline, the distribution of the rows is uniform, the first row is used for the baseline numbering, the second row represents the timeline and includes information about the date and hour of creation of a revision, the horizontal blue lines with arrows on both ends emphasize an individual day and the vertical blue lines indicate the end of one day; the dark blue small lines in between the parallel vertical blue lines point out the absence of work, the rounded rectangular nodes are used to emphasize the creation of branches and the orange line connecting the blue ovals outline the main code version. This sketch allows us to appreciate all the baselines and revisions of the item at a glance, as well as the relationships among baselines and the hierarchical association between baselines and revisions.

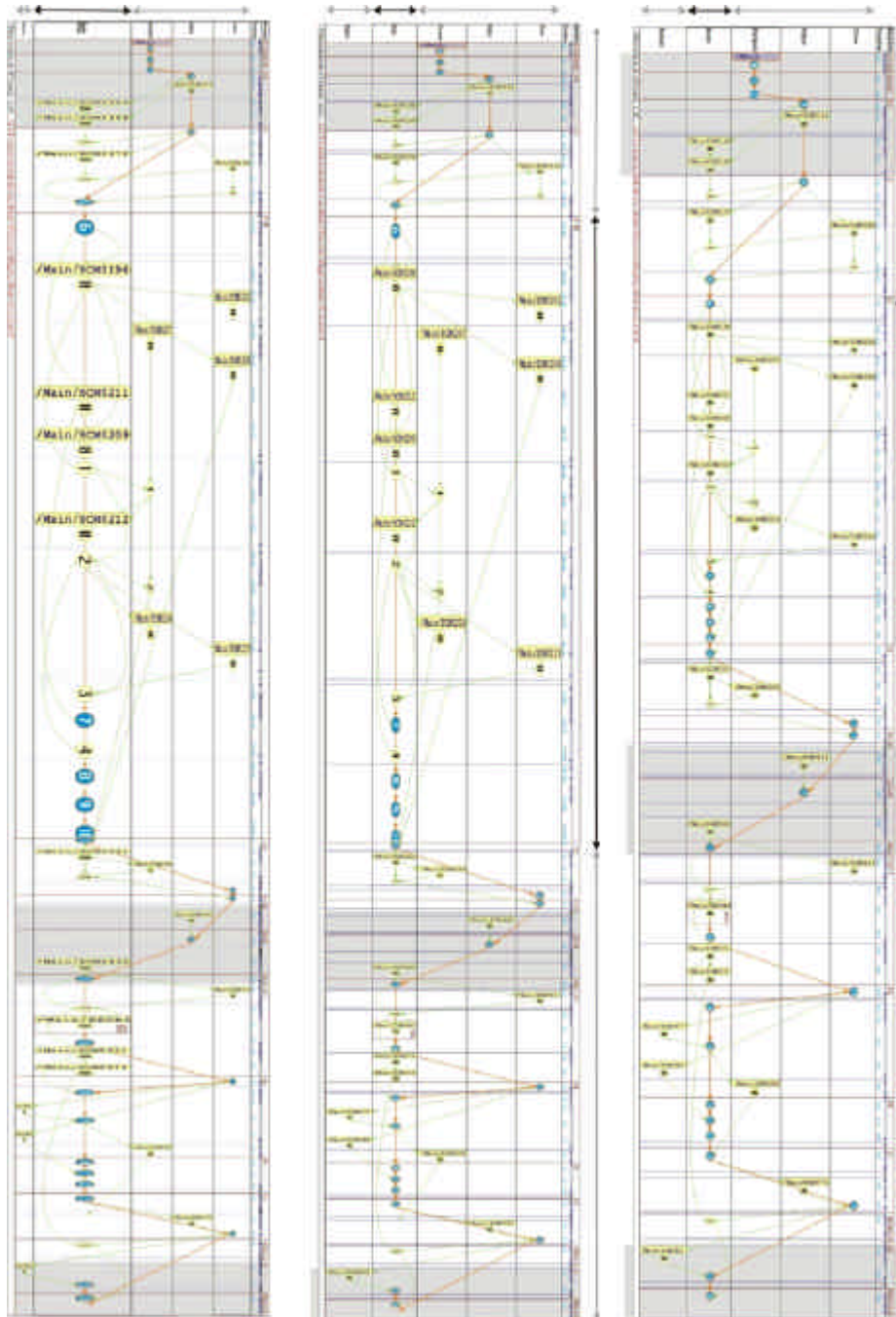
The second and third sketches show the use of the bifocal display expanding the column of the twelfth baseline and the row corresponding to Borja, while the information of the other baselines shrinks, keeping in the screen all the versioning information of the item and allowing us to concentrate on the area in which the item has more activity; although it is also possible to focus on several points of the representation at a time using the polyfocal feature of the solution.

An interesting key point of this representation (available on figure 4) is the use of dark blue small vertical lines to exhibit times when a day has not produced any changes in the item (as it would usually happen during weekends); this can be seen for the weekend of 25<sup>th</sup> and 26<sup>th</sup> of February, which is a normal situation, but also for the period between the 2<sup>nd</sup> and the 7<sup>th</sup> of March, which may be interesting for the project manager to observe. This simple approach allows us to discover intuitively these “non-working” or stable periods. The same approach is used when the period of stability covers a whole month; in that case, a small, blue circle is used to represent a stable month in the timeline and the white-gray switching is maintained (this situation can be seen in figure 3, where April and May produced no changes within baselines 14-22).

Furthermore, the proposal supports more interactivity; the users can select the main branch or regular branches of the application as a means of uncluttering complex revision trees; as a result, the application will bring out all its first level associations along the presentation, they can also select any node to highlight its connections. By means of the interaction, the representation can be modified in order to show the user the exact information he/she wants to see.

One of these interaction techniques in the proposed visualization is a focus + context technique: the use of variable width columns depending on the number of revisions in the baselines and the use of bifocal and polyfocal displays. The bifocal display consists of the capacity of the visualization to expand the rows and columns intersected in the area of interest; the polyfocal display has the same distortion behavior but allows focusing on more than one area.

On the other hand, filtering is another useful possibility: it may be interesting to have the same representation but for a particular period of time, or including only the information of selected developers.



**Fig. 3.** Representation proposed for Revision Trees (three different situations depending on the interaction are shown).

We recommend reviewing all these design details carefully on figure 4 and use table 1 for a listing and description of the variables and visual elements in the representation.

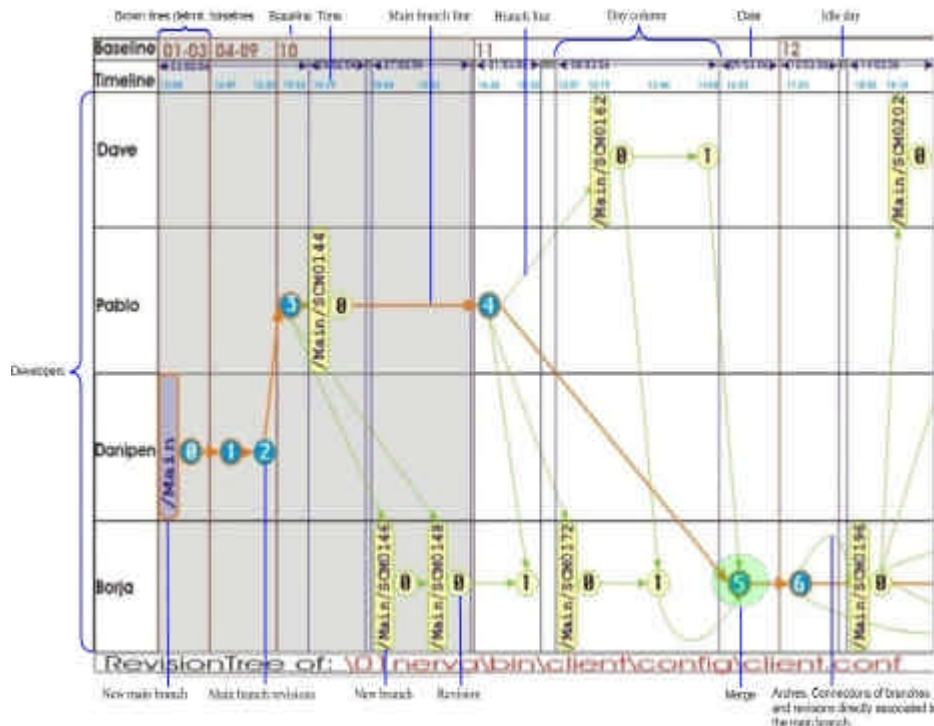
**Table 1.** Visual elements and variables represented on the Revision Tree visualization

Visual element	Description	Representation
Authors	Names of the developers.	Label with the name of the developer.
Baseline	Number of the baselines.	It is displayed at the top of the visualization.
Date	It indicates the creation date of branches, baselines or revisions.	Label with the date. It is exhibited on the timeline.
Day column	It is the graphical space for the representation of a day having activity in the creation of branches, baselines or revisions.	A dark blue line with arrows on both ends.
Time	Shows the time when a new branch or revision has been created in the main branch or any other.	Label with the time. It is shown on the timeline.
New main branch	Indicates the creation of the main branch.	Purple large oval
New branch	It shows the creation of a new branch.	Yellow large oval
Main branch line	It highlights the main branch.	orange arrows
Arches	Connect the branches and revisions created by the developer working on the main branch.	Green arches.
Main branch revisions	Revisions created in the main branch.	Blue nodes
Branch line	The branch line connects the main branch with other branches and the revisions within that branch or between two branches.	Green arrows.
Revision	This symbol represents the creation of a new revision of the software item.	Yellow nodes
Merge	A merge occurs when one or more branches are combined with the branch.	Incoming arrows coming from other branches into the main branch.
Idle day	It denotes a day without any activity in the creation of baselines, branches or revisions.	A small vertical blue line in the timeline, for each day with no activity.
Idle month	This symbol represents an entire month without any activity in the creation of baselines, branches or revisions.	A small blue circle is used to represent a stable month in the timeline

When assessing the proposed Revision Tree, it becomes evident that it is possible to obtain a great amount of information at a glance and that a detailed explanation to discover data of relevance is not required; it is easy to follow up on contributions to the development of a software item and understand how it has evolved throughout. This visualization also provides useful information for project managers; they can become aware of who has been working most in the development of the item, if someone has quit or been fired from the company, as well as discover if the last revisions made by that programmer to the item were merged or if there is a merge that has never been done for any other reason. They can also get information about the periods with more activity in the component and recognize when the item is stable, due to the fact that it is not suffering frequent changes. At present, it is possible to get



a lot more information through the careful checking of every visualization detail, particularly if a large real-life set of data is used.



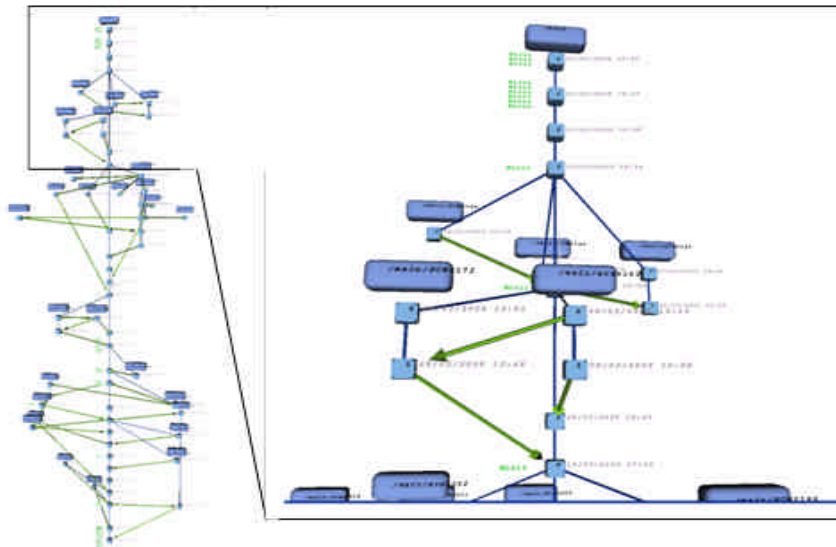
**Fig. 4.** Revision Tree layout showing all design features and the use of the bifocal display.

#### 4 Case study: Advantages of the Revision Trees as compared with the use of 3D revision trees

The development of visual representations using three dimensions has become popular during the last few years. We used the 2D Revision Tree representation to visualize a large revision tree and compared the results with the ones produced by Códice Software's PlasticSCM and its visualization 3D version tree tool, VRCS and Perforce. Figure 5 shows the 3D version tree for the same example as the one shown in figure 3 and a zoom in for the first 11 baselines.

The 3D version tree is eye-catching; it has a line representing the main line of the development and green arrows showing the merge of revisions. Along with the nodes, there are labels indicating some information about the baselines and revisions. Although the representation is visually appealing, it shows a number of drawbacks, the first one being that it is a static representation: the user can only change the point of view or choose how far he/she is looking at it (i.e, it is only possible to turn around

the tree and zoom into a region to get closer to a node or area). When zooming in, the size of the node increases and it is harder to manipulate the tree and the context is lost because the visualization lacks a context + focus view, so the user becomes disoriented; when we zoom out, the tree becomes a 3D shape with no special SCM meaning. Moreover, even after zooming in, you cannot see all the information represented due to occlusion; the front nodes hide the other nodes representing revisions.



**Fig. 5.** Sample three dimensional version tree produced by PlasticSCM showing the evolution of an item and a zoom in on the selected area.

Although the 2D revision tree performs better in a large screen, even a small space such as the one used in figure 3 (1 third of a page in a colored print out) can help the user to obtain a general idea of what was the evolution of a particular item.

At this point, the work of Ware [23] has been very helpful: he analyzes the use of 3D on information visualization and proposes the use of a 2 1/2D attitude when designing representations. It basically addresses many of the drawbacks of 3D visualizations and suggests the use of 3D consciously in combination with 2D for producing better visualization solutions.

Before going on with our analysis, we will discuss some details VRCS and the Perforce visualization tool. Our main concerns with the visualizations presented in [19][20] and illustrated in figure 1 are the lack of a focus + context view, the navigation through the structure, and how it can behave with the presentation of complex systems due to the high processor and memory demands of three dimensional visualizations and occlusions. As we discussed above, the visualization of large revision histories for one item using the three dimensional version trees has some limitations. Therefore, the visualization of large repositories with many items containing lots of baselines and revisions would result in a very large hard to navigate

visualization and probably would not provide, within a short time, the information required by the user.

Besides, the representation produced by Perforce (figure 2) offers an overview + detail approach, loosing this way a great amount of screen real state. It shows information about branching and merges and it is possible to obtain the date and time of revisions by clicking over the nodes and reviewing the information on the Details tab on the left panel. However, it does not provide information about the programmers contributing to the development of the item, how long the developers have been working on the item, nor in regards to periods without activity; furthermore, it is not possible to compare two baselines or see the time line at a first sight. In conclusion, this visualization is static and does not offer interaction options.

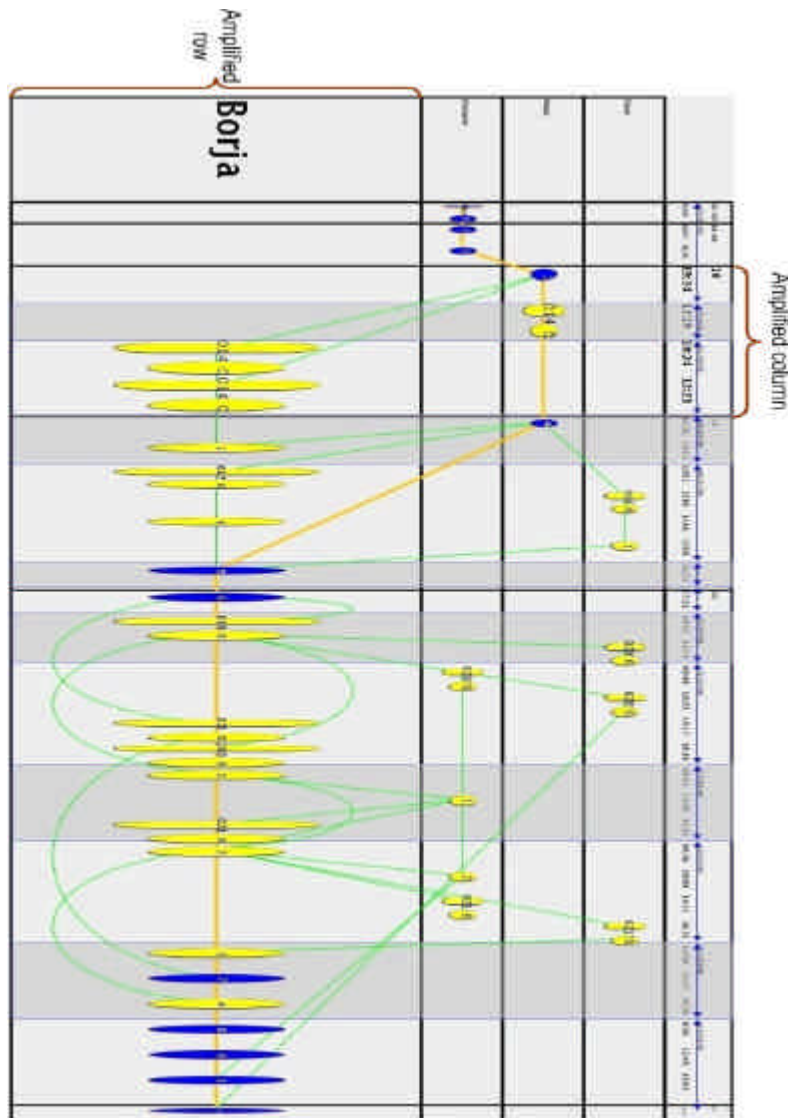
**Table 2.** Comparison of visualization tools for the representation of revision trees.

Questions	PlasticSCM	VRCS	Perforce	Revision Tree
Does the visualization provide a focus + context view?			X	X
How many developers are participating in the development of the software item?				X
Who are the developers contributing to the evolution?				X
Who is the programmer with more contributions to the evolution of the item?				X
How many baselines constitute the whole evolution process?	X	X		X
Does the tool offer information about dates and times of the creation of baselines and revisions?			X	X
Is there a revision without been merged after a long time?			X	X
How long has been the development of the item?	X	X	X	X
Which baseline has more branches and revisions?				X
Which branch has more modification activity?			X	X
Which is the period of time that does not show any activity?				X
Is there a period when the item was stable and then suddenly started having a lot of activity?				X
Is it possible to compare baseline activity?				X

The table 2 presents a list of questions to compare the visualization tools discussed above with our design, showing an X mark when the tool demonstrate evidence of answering a question within a short time and little effort while visualizing a large version tree.

Currently, in order to show the validity of our proposal, we have implemented a prototype that features the main ideas exposed above. An incomplete evolution corresponding to the first twelve baselines of the software item, used as an example,

is shown in figure 6. It is easy to realize that Borja has done many contributions on versions 10, 11 and 12 (it was rotated due to page constrains). Revision 1 pointed out by the node 4 in the main development line has not been merged, the development of the item started on February, 23rd and revision no. 11 was reached 15 days later; the baseline with more branches and revisions is number 12; branches 0162 and 0172 are tied into the number of revisions. There are two periods of time in which there is no activity in the item and whose dates are between February 24th - February 27th and March 1st - March 8th, also, the comparison between the baselines is immediate in this case, due to the short period of time under consideration.



**Fig. 6.** Revision tree of the first twelve baselines of the overall evolution example.

The brackets on the figure 6 highlight the use of bifocal display, the row corresponding to Borja has been amplified as well as the column associated to baseline 10, to show specific details. This figure also shows the use of a focus + context view; it allows getting information about specific areas while showing the general picture.

We consider that, after doing this comparison, our proposal is a valid representation for the visualization of revision trees and its complete development incorporating the features described in this paper as well as others that will result from its evaluation by users, and which will end in a powerful visualization easy to use and learn while providing the information needed by the users.

## 5 Conclusions

Revision Tree provides a focus + context view, a grid structure to which all programmers are familiar with, a timeline to guide and position users in time and space and several interaction possibilities to make the information the user needs available. With this presentation, the user can get many answers about how the evolution of the item is progressing and the team is always aware about who is working on the different baselines and revisions.

Whereas the visualization is always visible for all revisions, the users can review all the baselines and revisions in a very short period of time, therefore, there is no hidden information or an occlusion.

The timeline representation is clear, showing the complete time interval since the item was created, it also supports temporal comparisons and it made the concurrency of the programmers evident. Besides, the interactivity adds functionality to filter or focus on specific areas: in synthesis, the two-dimensional visualization offers a clear and functional presentation.

On the other hand, the three dimensional visualization does not support a focus + context view and only provides useful information at the detail level; to get the required information the user has to interact with the system for a long period of time. It allows the inspection of only one revision at a time, while nodes and labels could occlude the presentation at a general level.

The visualization presented in this paper shows enough evidence to state that for the representation of the evolution and collaboration in the development of software items a two dimensional representation offering several interaction possibilities can result in a powerful solution for the visualization of multidimensional data.

Future improvements will consider the improvement of interaction techniques, the use of linked views for the visualization of the directory structure of baselines and the comparison of several baseline structures. Furthermore, a usability test to compare the three dimensional representation and Revision Tree will be conducted.

There are other challenges also related with this problem; the representation of the collaboration between programmers and all the items stored in the repository of SCM tools.

## 6 References

1. Estublier, J. *Software Configuration Management: A Roadmap*. ACM Press. (2000).
2. Berlack, H., Updike-Rumley, M. *IEEE standard for software configuration Manager plans*. (1990).
3. Theron R. Hierarchical-temporal Data Visualization using a Tree-ring Metaphor Lecture Notes in Computer Science. *Smart Graphics*, vol. 4073, Springer-Verlag, Germany (2006) 70–81.
4. Mackinlay, J. D., Robertson, G. G., and Card, S. K. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, (1991). 173–176.
5. Leung, Y. and Apperley, M. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, Volume 1, Number 2 (1994) pages 126–160.
6. Spence, R. *Information Visualization*. ACM Press (2000).
7. Card, S. K., Mackinlay, J., and Shneiderman, B. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufman (1999).
8. Storey, M. D., Cubranic, D., German, D. M.. On the use of visualization to support awareness of human activities in software development: a survey and a framework. *Proceedings of the 2005 ACM symposium on Software visualization 2005*, St. Louis, Missouri May 14-15, 2005 193–202.
9. D. Gracanin, K. Matkovic, and M. Eltoweissy, “Software visualization,” *Innovations in Systems and Software Engineering: A NASA Journal*, vol. 1 (2005) 221-230.
10. Xie, X., Poshyvanyk, D., and Marcus, A. (2006). Visualization of cvs repository information. In *WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 231–242, Washington, DC, USA. IEEE Computer Society.
11. Eick, S. G., Graves, T. L., Karr, A. F., Mockus, A., and Schuster, P. Visualizing software changes. *IEEE Trans. Softw. Eng.*, (2002) 28(4):396–412.
12. Voinea, L. and Telea, A. (2006). An open framework for cvs repository querying, analysis and visualization. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 33–39, New York, NY, USA. ACM Press.
13. Voinea, L. and Telea, A. (2006). Mining software repositories with cvsgrab. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 167–168, New York, NY, USA. ACM Press.
14. Voinea, L. and Telea, A. (2006b). Multiscale and multivariate visualizations of software evolution. In *SOFTVIS 2006*. Association for Computing Machinery Inc.
15. Gall, H., Jazayeri, M., and Riva, C. (1999). Visualizing software release histories: The use of color and third dimension. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, page 99, Washington, DC, USA. IEEE Computer Society.
16. Morris, S.A., Yen, G., Wu, Z., Asnake, B.: Time line visualization of research fronts. *Journal of the American Society for Information Science and Technology* 54 (2003) 413–422.
17. Card, S., Suh, B., Pendleton, B., Heer, J., Bodnar, J. *TimeTree: Exploring Time Changing Hierarchies*. IEEE Symposium on Visual Analytics Science and Technology (2006).
18. Kumar, G., Garland, M. *Visual Exploration of Complex Time-Varying Graphs*. IEEE Transactions on Visualization and Computer Graphics, Vol. 12, Num. 5 (2006)
19. Lanza, M. The evolution matrix: recovering software evolution using software visualization techniques. *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*. (2001). 37–42. New York, NY, USA. ACM Press. ISBN 1-58113-508-4.
20. Koike, H. The role of another spatial dimension in software visualization. *ACM Trans. on Information Systems*. 11(3):266-286. (1993).
21. Koike, H., Chu, H.-C. *VRCS: Integrating version control and module management using interactive 3d graphics*. In *VL '97: Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)* Washington, DC, USA. IEEE Computer Society. (1997).
22. Rao, R. and Card, S. K. (1994). The table lens: merging graphical and symbolic representations interactive focus + context visualization for tabular information. In *CHI '94: Proceedings of SIGCHI conference on Human factors in computing systems*, pages 318–322, New York, NY, USA. ACM Press.
23. Ware, C. (2001) *Designing with a 2 1/2D Attitude*. *Information Design Journal* 10(3) 255-262.