Supporting the understanding of the evolution of open source software items

Roberto Theron[*]
Antonio Gonzalez[†]
Francisco J. Garcia[‡]
Department of Computer Science
University of Salamanca, Spain

Abstract

This paper intends to support the awareness of open source software developers and project managers through the understanding of the evolution of software items.

We propose a simple interactive visualization tool that focuses in the representation of the collaboration that takes place between developers during the development of software items. The visualization, which is supported by several interaction techniques, presents detailed information on a single software item regarding the creation of baselines, branches and revisions, and useful date and time details for the arrangement of the development time line and collaboration representation. It also allows the filtering of dates using dates ranges, the hiding of rows and columns and the review of each revision log.

To demonstrate the usefulness of our proposal we performed a case study extracting the evolution details from a software item of XBMC(formerly known as "XBox Media Center"), an open source software project under the management of SourceForge.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Screen design; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: radiosity, global illumination, constant time

1 Introduction

The open source software community has become of increasingly importance and size the last decade, and the number of researchers studying its development and contributing with tools and ideas have also increase. At the same time many online software repositories, such as SourceForge, Apache, Google Code and Freshmeat, have come into scene to support the development of open source software projects. This has opened a space for the concurrent contribution of many developers distributed across the world.

The collaboration between developers in open source software projects is crucial and demands a lot of informal communication and coordination. Our intend is to support the awareness of the project managers and developers regarding the evolution of items and the collaboration taking place on its development.

Currently, there are many commercials tools that support the versioning of code and configuration management. Even though CVS and Subversion remain the most widely used tools by development companies and contributors of open source projects. According to the definition of [Estublier 2000] Software Configuration Management enhance the environment of the developers, managing concurrency and collaboration, and recording changes including time, date, which modules were affected, how long the modification took and information about who did the change. The Software Configuration Management tools support many other functions, but the above functions are shared by most code versioning tools; such as Subversion [Collins-Sussman et al. 2004]. Hence the importance of software 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 there is an important lack of mechanisms to convey by means of proper representations of how the contribution and collaboration of team members occurs in a particular project.

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. However, Software Configuration Management tools and code versioning tools still can be enhanced by using highly interactive visualizations rather than mere static representations.

In recent years the field of information visualization has played an important role 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 [Theron 2006].

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 [Mackinlay et al. 1991]. Additionally, we take into consideration several techniques to support navigation, interpretation of visual elements and understanding relationships among items in their full context [Leung and Apperlley 1994].

There are also many information visualization techniques, each one with its advantages and disadvantages, so frequently it is required to use a sort of combination to provide a real solution to end users. Spence [Spence 2000] and Card [Card et al. 1999] provide excellent surveys of information visualization mechanisms and techniques. We support our visualization through the use of a grid based structure, selection, navigation, filtering, polyfocal display, a tree hierarchy (a directed graph) for representing revisions, a focus + context view and a time line as visualization techniques.

We considered in our design what is going on in the project, who

[*]theron@usal.es
[†]agtorres@usal.es
[‡]fgarcia@usal.es

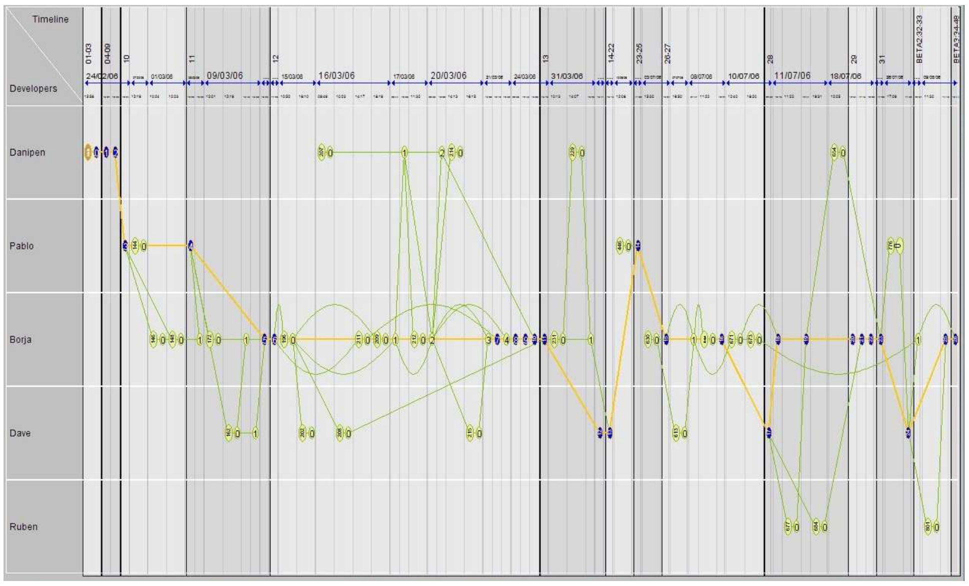


Figure 1: *Normal View of the main visualization panel.*

else is working on the project, what are they doing, how long have they been working on a revision, how their work may impact the work of others, if developers are collaborate between them or work separate from each other and the overall framework designed by Storey in [Storey et al. 2005] for describing the visualizations of human activities in software engineering.

This paper is committed to present a 2D interactive visualization tool, 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 our visualization; Section 4 is a case study in which the tool is used to review open source software items, and finally, Section 5 discusses the conclusions.

2 Related Work

Considerable work has been dedicated to study the software visualization and information visualization areas. Although we concentrate on the evolution of individual items and the collaboration of software teams on its development, in this section we review some useful ideas that have been applied in the visual representation of code versioning tools repositories.

As a starting point it is useful to consider the definition proposed by Gracanin [Gracanin et al. 1994] who 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 the scope and content of the visualization, who will be the audience, 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.

On the other hand Xie [Xie et al. 2006] list a set of questions that can be used to guide the design of visualizations of open source software repositories; for the purposes of this paper it is relevant determining which authors work on the same file, when was a modification made and how many authors worked on the release of the system.

Moreover, Eick [Eick et al. 2002] accurately states that a fundamental problem in visualizing software changes is to choose effective visual representations or metaphors and then 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. We consider that the use of effective visual representations and metaphors in combination with interaction techniques are a powerful combination for making the visualization display different perspectives of the representation, and thus useful in the representation of the evolution of software items.

Many authors, like Voinea and Telea [Voinea and Telea 2006c] support the idea that software configuration management repositories are valuable for project accounting, development audits and understand the evolution of software projects. We strongly agree about the richness of software repositories; so the effective design of the repository of Software Configuration Management and Code Versioning tools can provide information about the development process that is not possible to acquire from any other source. We also consider that through a well design visualization its possible to navigate the repository data and get an insight of what is going on in the project. The same authors, Voinea and Telea, also propose two visualizations for software management configuration repositories in [Voinea and Telea 2006a] and [Voinea and Telea 2006b]. 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. At the moment, we do not incorporate textures but we include the use of colors for the identification of revisions and branches.

Other important contribution to the software evolution field has be done by Gall [Gall et al. 1999] who 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

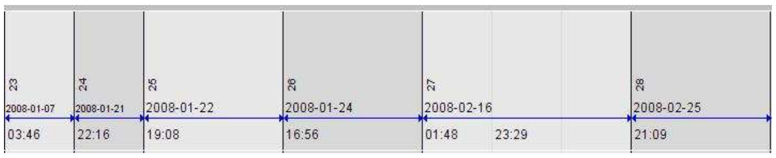
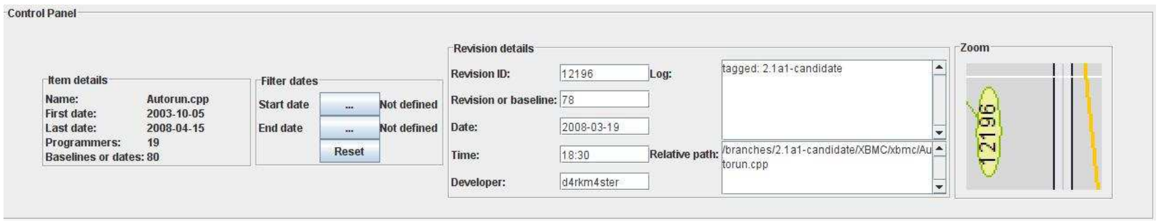


Figure 2: *View of the time line design.*



Control Panel

Item details

Name:	Autorun.cpp
First date:	2003-10-05
Last date:	2008-04-15
Programmers:	19
Baselines or dates:	80

Filter dates

Start date	...	Not defined
End date	...	Not defined
	Reset	

Revision details

Revision ID:	12196
Revision or baseline:	78
Date:	2008-03-19
Time:	18:30
Developer:	d4rkm4ster
Log:	tagged: 2.1a1-candidate
Relative path:	/branches/2.1a1-candidate/XBMC/xbmc/Autorun.cpp

Zoom

Figure 3: *Control Panel of the visualization tool.*

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

There are also several proposals addressing the representation of temporal spaces using many different structures. Morris [Morris et al. 2003] 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 [Card et al. 2006] developed a visualization that allows exploring hierarchies that change with time using searches, navigating through a hierarchical presentation and filtering results with the assistance of a time slider control. Theron [Theron 2006] 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 [Kumar and Garland 2006] 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. We decided to use a linear time line to represent more intuitively the relationship between programmers, revisions, baselines, dates and times.

One of the representations that have strongly influence the design of our visualization, because of the disposition of its elements, is the one presented by Lanza in [Lanza 2001]. This proposal deals with the visualization of software attributes throughout the time using an evolution matrix with variable size rectangular 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 [Gall et al. 1999].

Other related visualization has been developed by Koike [Koike and Chu 1997] describing a representation that shows the evolution of items from the repository of the software management configuration tool. On this visualization each software item is represented using two dimensions and the overall visualization with three dimensions. This representation is a classic reference in the evolution field and we consider it very interesting. However, we are in favour of the use of 2D visualizations supported by several interaction techniques, because they could result more effective while visualizing large data sets.

Finally, the Revision Tree is a visualization proposal presented in [Theron et al. 2007] for the representation of the evolution of software items. We based this proposal in that paper and have expanded its scope to represent the evolution of open source software item, we also have included several new interaction techniques to make the visualization more usable and intuitive, and developed a tool for extracting open source software repositories details.

3 Visualization of the evolution of open source software items

In this section we propose a 2D representation for the collaboration history of software items and baseline structures. The visualization is applicable to open source software projects and commercial products, as well as to repositories managed by tools such as CVS and Subversion, or any commercial tool.

The visualization presented here 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.

Before going on, it is fundamental to highlight that a baseline is a label that references the software status in a particular moment, and it could be present or not depending on the tool used in the management of the software repository. So, in the cases where the baseline label is not present we map the dates into a sequential number for our visualization tool.

The following subsections discuss the design details of our proposal, the time line and the possibilities that offer the control panel of the application.

3.1 Visualization design

The problem at hand presented several challenges that were addressed in the proposed visualization: the representation of large revision trees where the software items have several branches and each branch many item revisions; the navigation through the version tree offering a focus + context view; support to interactivity to enable the inspection of many revisions at a time, exhibit the collaboration of developers for every branch and correlate all the information with the time line. The main components of the resulting

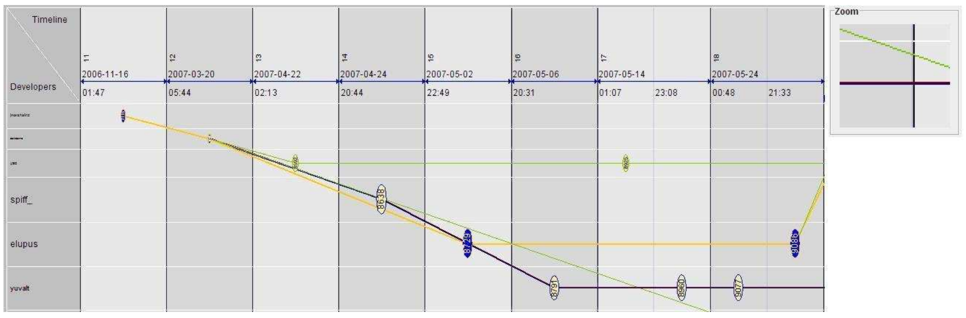


Figure 4: *Segment of the evolution of a software item.*

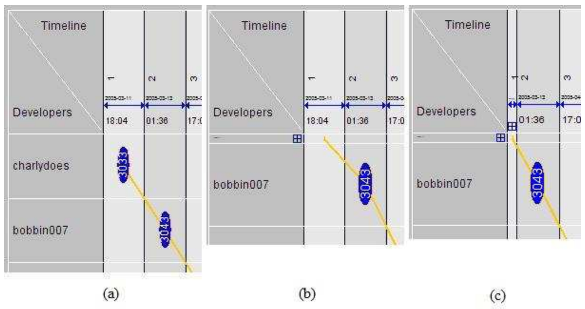


Figure 5: *Hiding of rows and columns.*

visualization are the time line, the control panel and the visualization of the revision tree.

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 developers, baselines and revisions; using the rows to represent the authors and the columns for the elements of the time line. Moreover, grids and matrices 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.

The figure 1 exposes the normal vista of the design; it uses a time line for variable width columns to accommodate the revisions in each baseline or date (depending of the type of repository), the distribution of the rows is uniform for the authors while the first line is used to arrange the time line. The cells of the grid contains the revisions, which are aligned with the time line according to its baseline, date and time.

3.2 Time Line

The time line (see figure 2) indicates the baseline numbering on the center, and the date and hour of creation of a revision in the bottom. The horizontal blue lines with arrows in both ends emphasizes an individual day, the vertical black lines indicate the end of one day and the vertical light gray lines delimits a specific activity within a given time (review carefully the baseline 27). The vertices of the revision tree are carefully aligned with the elements of the time line to show the evolution of the software item.

3.3 Control Panel

This area of the visualization complements the interaction of the user with the visualization, and the user has the possibility of choosing when to show or hide this control. The control panel is shown in the figure 3, and it is composed by the panels item details, filter dates, revision details and zoom. The panel item details displays the name of the software item, its creation date, the last date when it was updated, the number of programmers participating in its evolution, and the number of baselines or dates of its evolution. The filter dates panel allows to filter the presentation by ranges of dates, while revision details panel presents the details about a selected revision and and the zoom panel zooms the area where the mouse is located. The filtering of dates is useful for reviewing a specific period of time and get a more specific view of an interesting activity range. Consequently, the Revision Details panel provides even more details about a selected revision, including the log that have been entered for that revision and the path of the revision in the repository. Finally, the Zoom panel focuses in the position where the mouse is located and amplify by a factor of 2 that area.

3.4 Interacting with the revision tree

The revision tree is our proposal for representing the evolution and collaboration during the development of software items. It layouts the revisions, baselines and branches accordingly, based on the time line information and the developers working on revisions and branches. The position of the revisions is the intersection between the row of a programmer and the column representing a specific time in the time line. The revisions are represented by ovals and the revision number is aligned horizontally if it has 1 digit and vertically if it has more than 1 digit. The blue ovals are revisions within the main branch and the others are revisions within branches. The orange line connecting the blue ovals delineate the main code versioning and the green line the branches. However, when there are duplicated branches, where the revisions belongs to more than one branch, the line connecting revisions is composed by more than one color; where each color represents a specific branch. This representation let us appreciate all the baselines and revisions at a glance as well as the relationships among baselines and the hierarchical

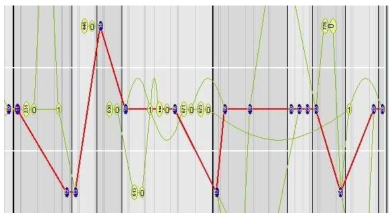


Figure 6: *Highlighting of the remaining path of the main branch.*

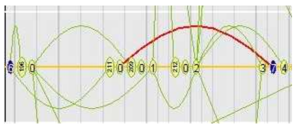


Figure 7: *Highlighting of revision created by a branch in the same path that the main branch.*

association between branches and revisions. The figure 4 shows a piece of the visualization where can see the color coding and two branches sharing some most of their revisions; the small square on the top right of the figure shows a zoom of a segment of this branches and allows to observe two different colors, one for each branch.

Moreover, the visualization supports many interaction techniques to allow discover information not visible at first glance and generate new visualizations perspectives. The figure 5a shows a normal view, the figure 5b shows the hiding of one developer and the figure 5c shows the hiding of one developer and one baseline. Additionally, it supports polyfocal displays allowing to expand rows and columns in many different points of the visualization, and offers the possibility of exchanging rows, as will be shown in the case study.

The visualization provides the possibility to select the branches to highlight its path. In the case of the main branch, when it is selected at some point of the visualization the remaining path of the branch is highlighted, as shown in figure 6, and in the case of regular branches the remaining path is highlighted until it is merge in the main branch. This feature is even more valuable when there is a branch in the same path that follows the main branch and it is required to highlight the connection between revisions or the merge of one of these branches, figure 7 shows an example of the last case.

4 Case Study: Exploring the evolution of software items

When assessing our proposal it stands out that it is possible to obtain a great amount of information at a glance and it does not require a detailed explanation to discover data of relevance; it is easy to follow up 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 realize who have been working most in the development of the item, if someone has quit working for the project they can 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 can also recognize when the item is stable because it is not suffering frequent changes. Actually, it is possible to get a lot more information through the careful checking of every visualization detail and especially if a large real life dataset is used. Although the 2D revision tree performs better in a large screen, even a small space such as the one used in figure 1 (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.

The following questions were used to assess our visualization and improve its weaknesses. We consider that the proposed visualization can answer these questions quickly; some of them at first glance while others questions could be answered with little effort and some interaction. To demonstrate this, in this section we use one software item taken from XBMC (formerly known as ”XBox Media Center”), and open source project.

1. Does the visualization provide a focus + context view?
2. How many developers are participating in the development of the software item?
3. Who are the developers contributing to the evolution?
4. Who is the programmer with more contributions to the evolution of the item?
5. How many baselines constitute the whole evolution process?
6. Does the tool offer information about dates and times of the creation of baselines and revisions?
7. Is there a revision without been merged after a long time?
8. How long has been the development of the item?
9. Which baseline has more revisions?
10. Which branch does have more revisions activity?
11. Which is the period of time that does not have activity?
12. Is there a period when the item was stable and then suddenly started having a lot of activity?
13. Is it possible to compare baselines activity?

The complete evolution of a software item could be seen in the figure 8. We can answer immediately, without using the control panel information, how many developers are participating in the development of the software item, how many baselines constitutes the evolution, which revisions and branches have been merged. It is also possible to get information about what baseline, branch or date has more revisions, and it is possible to compare the activity of baselines and dates. The visualization also provides filtering mechanisms that are not assessed by the above questions, but those mechanisms allows to answer some questions which answers are not available at first sight. The figure 9 shows the results of

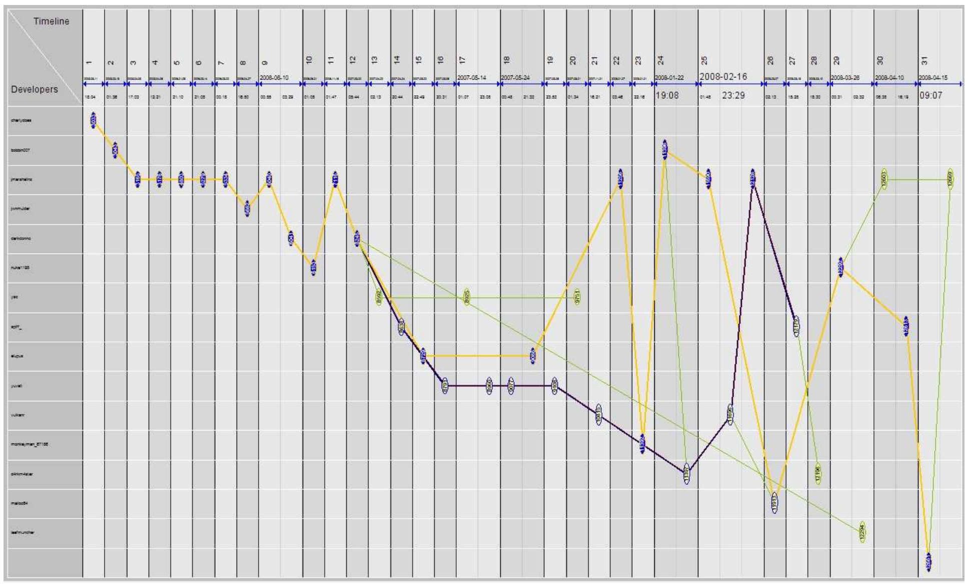


Figure 8: *Visualization of the evolution of a software item from XMBC; example 1.*

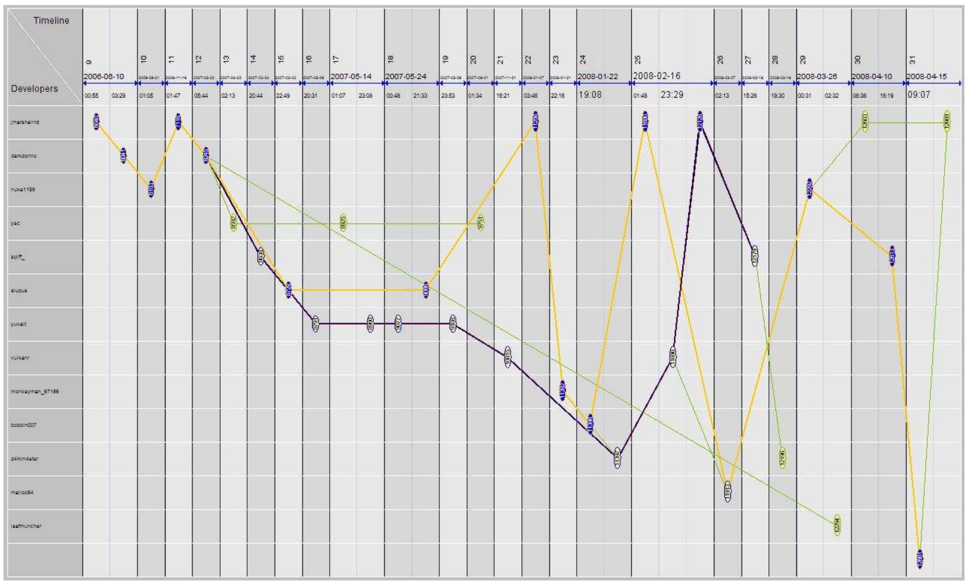


Figure 9: *Results of filtering dates in the software item.*

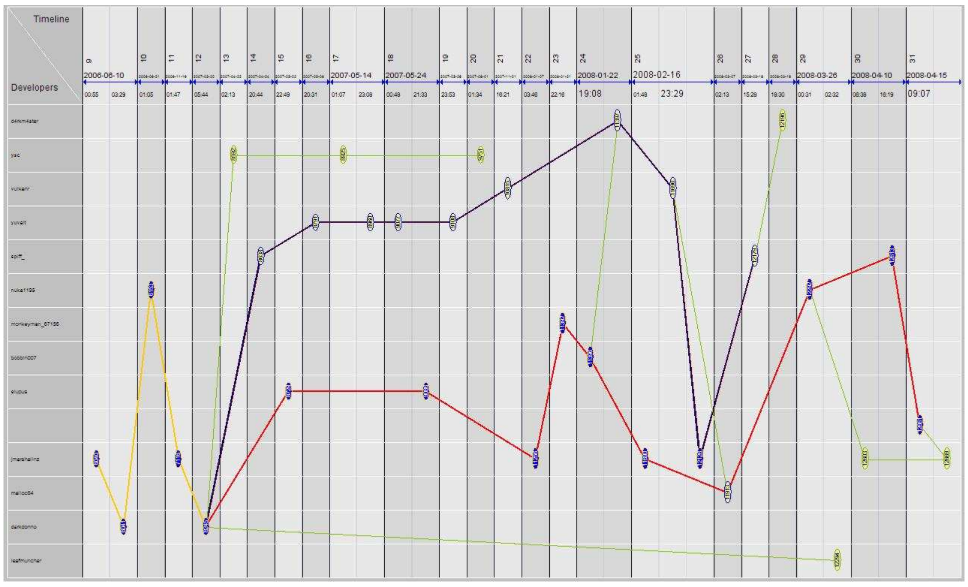
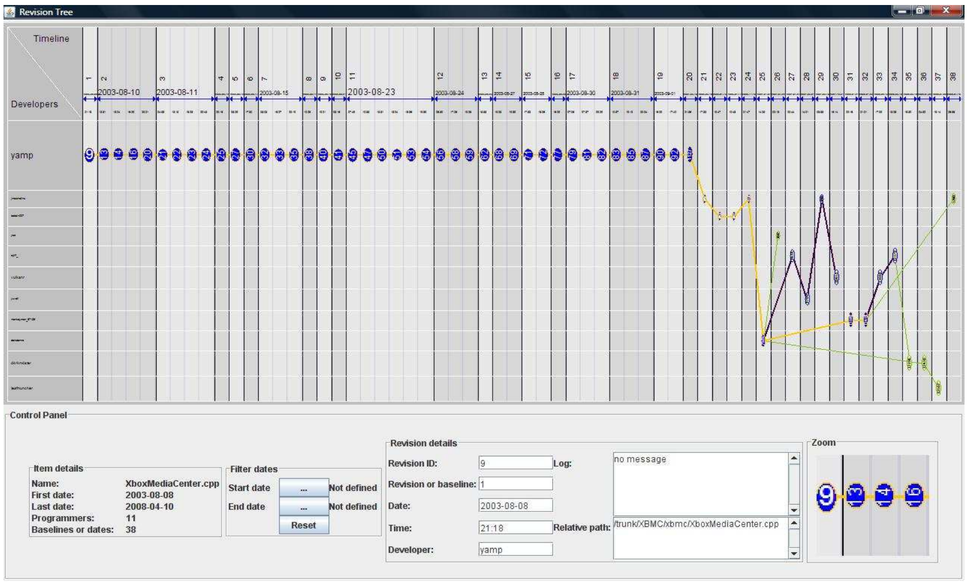


Figure 10: *Final results after exchanging rows and hiding rows and columns.*



Item details

Name: XboxMediaCenter.cpp
First date: 2003-08-08
Last date: 2008-04-10
Programmers: 11
Baselines or dates: 38

Filter dates

Start date ... Not defined
End date ... Not defined
Reset

Revision details

Revision ID: 9
Log: no message
Revision or baseline: 1
Date: 2003-08-08
Time: 21:18
Relative path: /trunk/XBMC/xbmc/XboxMediaCenter.cpp
Developer: yamp

Figure 11: *Evolution of* XboxMediaCenter.cpp*; example 2.*

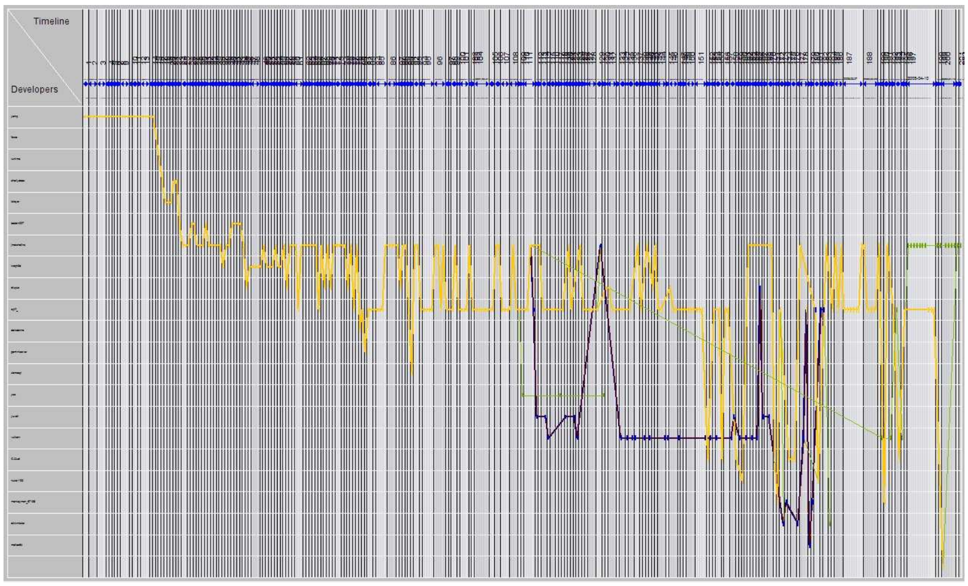


Figure 12: *Complete evolution of* VideoDatabase.cpp*; example 3.*

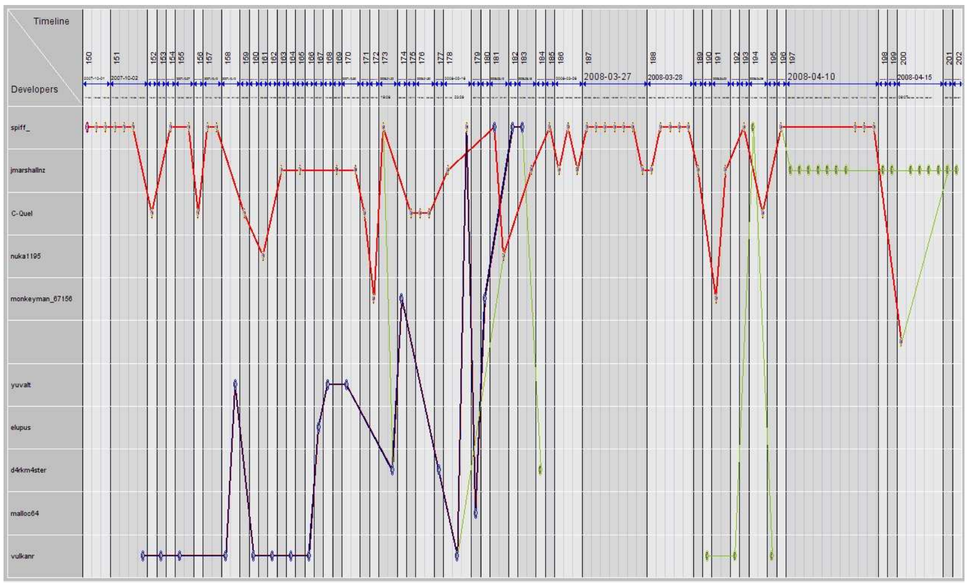


Figure 13: *Dates filtering applied to* VideoDatabase.cpp.

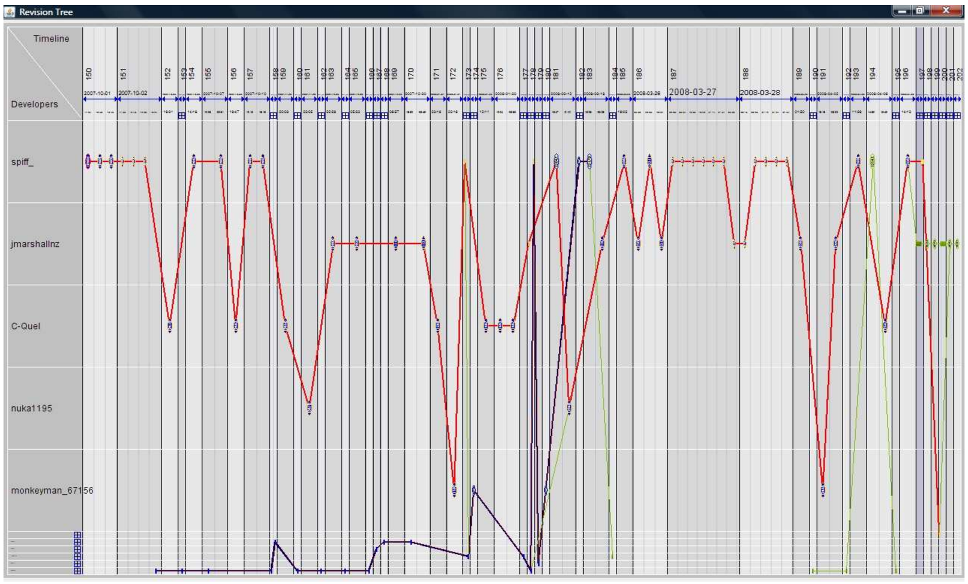


Figure 14: *Hiding of rows and columns.*

filtering dates in our example. Finally, we exchange some rows, then we hide some developers and baselines, expand some rows and columns and highlight the path of a revision. After doing this, we get a different view of the evolution of the software item, as shown in the figure 10. Now, we can see clearly the application of the polyfocal technique and get details about who are the programmers that have contributed more to the development of the item; we have expanded the rows with more revisions to discover that *jmarshallnz* and *yuvalt* are the developers with more contributions. Moreover, we are able to see when the selected branch, highlighted in red, is merged into the main branch. If we continue interacting with the visualization we can get the names of all the developers working in the software item with only some mouse clicks.

After carefully reviewing the figure 9, it is not possible to clearly see the dates and times of the creation of revisions, but after expanding some columns these details become visible. Besides, the information about how long has been the development of the software item, which is the period of time that does not have activity, when the activity of the item has become stable or reactivated, could be gather after carefully reviewing the visualization and interacting with it. Moreover, the Control panel provides many useful details that could be used to save interaction time and effort.

We consider that the visualization provides many details, some of them are evident while others require some interaction, but the importance of the contribution is the combination of visualization and interaction techniques to offer the information the users need. This visualization is very simple, intuitive and powerful for supporting developers and project managers in the development of open source software.

Now, we are going to review the example in the figure 11. This example is very simple, but it allows us to see that the programmer that has worked the most of the main branch is *yamp*, and after he stop working in the main branch others programmers have alternated working on this branch. Then, some new branches have been created and the work is distributed between the others programmers, except *yamp*. This information could result very valuable for the project managers to make conclusions associated to the software item.

Finally, the figure 12 shows the evolution of a very active software item. Its evolution has been expanded from January 19th, 2004 until April 16th, 2008 and it continues evolving. The evolution of this item accounts 202 dates, this means that it has had activity with one or more revision in 202 different days. Moreover, 22 programmers have been participating in the development of this software item. In spite of the large quantity of information displayed in the figure 12 it is possible to clearly distinguish the branches and the overall evolution. However, it is advisable to filter some dates out to review in detail the evolution of the software item. The figure 13 shows the results of filtering out the dates before October 1st, 2007, and exchanging some some rows to group the developers working on the main branch. But, we still does not have enough visibility of the the names of the developers, so we hide some rows and columns to produce other view of the representation; the result is shown on figure 14. Hence, we can expand or hide some columns to enlarge the revisions and review in more detail. It is important to highlight that the images shown here have been produced by an standard monitor of 1280 x 800 resolution for testing purposes, so with a larger screen the results can improve significantly.

5 Conclusion

Our proposal provides a focus + context view, a grid structure to which all programmers are familiar with, a time line to guide and position users in the time space, a control panel for filtering dates, display additional details and a zoom panel; and several interaction possibilities to make available the information the user needs. With this presentation the user can get many answers about how the evolution of the item is going on 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 time, and therefore there is no information hiding or occlusion. The time line representation is clear, showing the complete time interval since the item was created, supports temporal comparisons and made the concurrency of the programmers evident. Besides, the interactivity adds functionality to filter or focus in specific areas: in synthesis the two dimensional visualization offers a clear and functional presentation.

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.

References

CARD, S. K., MACKINLAY, J., AND SHNEIDERMAN, B. 1999. *Readings in Information Visualization: Using Vision to Think.*

CARD, S. K., SUH, B., PENDLETON, B. A., AND HEER, J. 2006. Timetree: exploring time changing hierarchies. In *IEEE Symposium on Visual Analytics Science and Technology 2006 (VAST 2006)*, IEEE Computer Society, Baltimore; MD; USA. Piscataway NJ.

COLLINS-SUSSMAN, B., FITZPATRICK, B., AND PILATO, M. 2004. *Version Control with Subversion*. Sebastopol, CA USA: O'Reilly Media, Inc. ISBN: 0-596-00448-6.

EICK, S. G., GRAVES, T. L., KARR, A. F., MOCKUS, A., AND SCHUSTER, P. 2002. Visualizing software changes. *IEEE Trans. Softw. Eng. 28*, 4, 396–412.

ESTUBLIER, J. 2000. Software configuration management: A roadmap. *The Future of Software Engineering*. ISBN 1-58113-253-0.

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*, IEEE Computer Society, Washington, DC, USA, 99.

GRACANIN, D., MATKOVIC, K., AND ELTOWEISSY, M. 1994. Software visualization. *Innovations in Systems and Software Engineering Volumen 1*, Number 3, 221 – 230.

KOIKE, H., AND CHU, H.-C. 1997. 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)*, IEEE Computer Society, Washington, DC, USA, 168.

KUMAR, G., AND GARLAND, M. 2006. Visual exploration of complex time-varying graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5, 805–812.

LANZA, M. 2001. The evolution matrix: recovering software evolution using software visualization techniques. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, ACM Press, New York, NY, USA, 37–42.

LEUNG, Y., AND APPERLLEY, M. 1994. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction Volumen 1*, Nmero 2 (Junio), 126 – 160.

MACKINLAY, J. D., ROBERTSON, G. G., AND CARD, S. K. 1991. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, 173–176.

MORRIS, S. A., YEN, G., WU, Z., AND ASNAKE, B. 2003. Time line visualization of research fronts. *J. Am. Soc. Inf. Sci. Technol. 54*, 5, 413–422.

SPENCE, R. 2000. Information visualization. *ACM Press*.

STOREY, M.-A. D., CUBRANIC, D., AND GERMAN, D. M. 2005. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, ACM Press, New York, NY, USA, 193–202.

THERON, R., GONZALEZ, A., GARCIA, F. J., AND SANTOS, P. 2007. The use of information visualization to support software configuration management. *Lecture Notes in Computer Science Volume 4663/2007*, 317–331.

THERON, R. 2006. Hierarchical-temporal data visualization using a ring tree metaphor. *Lecture Notes in Computer Science. Smart Graphics*.

VOINEA, L., AND TELEA, A. 2006. Mining software repositories with cvsgrab. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, ACM Press, New York, NY, USA, 167–168.

VOINEA, L., AND TELEA, A. 2006. Multiscale and multivariate visualizations of software evolution. In *SOFTVIS 2006*, Association for Computing Machinery Inc.

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*, ACM Press, New York, NY, USA, 33–39.

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)*, IEEE Computer Society, Washington, DC, USA, 231–242.