Fractal Figures: Visualizing Development Effort for CVS Entities

Marco D'Ambros and Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland

Harald Gall
s.e.a.l. - software evolution and architecture lab
University of Zurich, Switzerland

Abstract

Versioning systems such as CVS or Subversion exhibit a large potential to investigate the evolution of software systems. They are used to record the development steps of software systems as they make it possible to reconstruct the whole evolution of single files. However, they provide no good means to understand how much a certain file has been changed over time and by whom. In this paper we present an approach to visualize files using fractal figures, *which (1) convey the overall development effort, (2) illustrate the distribution of the effort among various developers, and (3) allow files to be categorized in terms of the distribution of the effort following* gestalt *principles. Our approach allows us to discover files of high development efforts in terms of team size and effort intensity of individual developers. The visualizations allow an analyst or a project manager to get first insights into team structures and code ownership principles. We have analyzed Mozilla as a case study and we show some of the recovered team development patterns in this paper as a validation of our approach.*

1. Introduction

Software versioning systems appeared more than thirty years ago and are widely used since more than two decades [13] [15] [5], especially in conjunction with the advent of open source software. Recording the history of a software system during its development allows reconstructing the original design intentions of the developers, as well as their subsequent variations in time. Versioning systems also ease team software development. The facilities given by versioning systems and the amount of data retrieved fostered the research field of software evolution [10], whose goal is to analyze the history of a software system and infer causes to its current problems, and possibly predict its future.

Visualization techniques were widely used to study the evolution of software systems [1, 2, 7, 12, 14, 16], but none of them is focused on development effort and authors.

The question we want to answer with this paper is whether we can efficiently convey (1) *how* a certain file has been modified over time (in terms of effort, *i.e.,* modifications), and (2) *who* participated in its development and to which extent.

In this paper we present *Fractal Figures,* a scalable visualization technique to understand the evolution of software entities with respect to the involved developers. Fractal Figures can visualize not only the basic CVS products, *i.e.,* files, but also abstracted products, *e.g.,* the containing directories or a complete directory subtree. Fractal Figures also convery *gestalt* impressions, based on which it is possible to categorize the represented artifacts.

All the results and the examples presented in the paper have been obtained by applying the presented visualization technique on the Mozilla [11] case study.

2. The TIMELINE VIEW

The questions we are trying to answer when considering a specific artifact in a software system (*e.g.,* a file) is (1) how much has this file been changed, (2) by whom, and (3) to which extent are certain developers responsible for its evolution.

A possible way to partially answer these questions is depicted in Figure 1: It shows a TIMELINE VIEW (time as the horizontal axis from left to right) where the visualized rectangles represent revisions on CVS products (*i.e.,* files). In this figure we see the evolution of 3 specific files between 2001 and 2003, the exact horizontal position is determined using the CVS commit time-stamp. The colors represent the different authors, *i.e.,* the person who committed the file revision to the CVS repository.

This timeline view conveys the following information: The product in the middle has many more revisions than the others and seems to have been changed by several developers. Still, it fails at giving us both a qualitative and quantitative impression about the development effort and distribution among developers for each CVS product. It is not clear how much work each developer spent on a particular product. Moreover, it does not scale well (*i.e.,* the author infor-

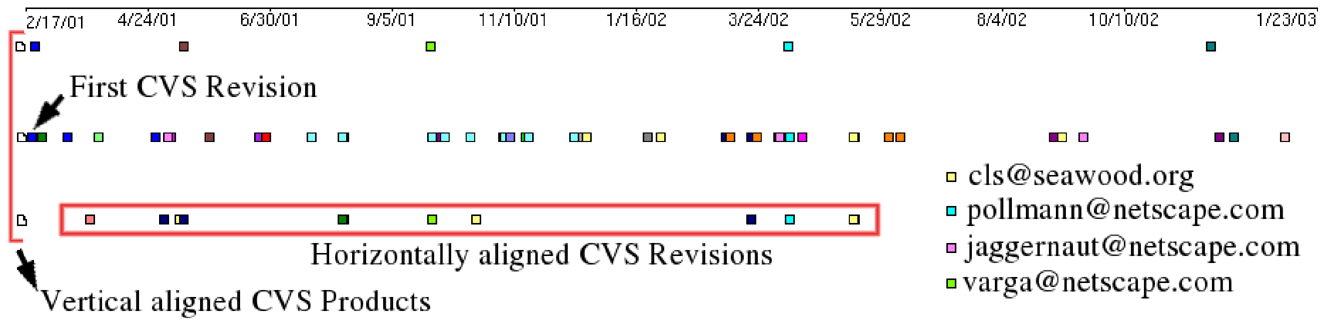

Figure 1. A TimeLine View applied to three CVS products of Mozilla

mation is quickly unreadable) if the number of products or revisions is high.

To have a scalable view we need to encapsulate all the author-related information belonging to a product (*i.e.*, a line in the TimeLine View) into one single figure: A *Fractal Figure*. This has the drawback of losing the time dimension, but we do not consider this as being a problem as in our tool implementation one can quickly navigate back and forth between the two views.

3. The Fractal View

A Fractal Figure (see Figure 2) gives an immediate view of how (in terms of development effort and distribution among authors) a product has been developed. We can easily figure out whether the development was done mainly by one author or whether many people contributed to it and in what intensity each contribution was.

The figure is composed of a set of rectangles having different sizes and colors. Each rectangle, and thus each color, is mapped to an author who worked on the product. The area of the rectangle is proportional to the percentage of check-ins performed by the author over the whole set of check-ins.

This visualization technique can be enriched by rendering a software metric measurement on the size of the figure (only one because the figure must be a square). This is useful for views containing many Fractal Figures.

3.1. Construction principles

As example we consider the *nsTextHelper.cpp* product of Mozilla. Table 1 shows the developers who worked on the product and, for each of them, the number of commits performed.

Figure 2 depicts the construction process of the corresponding Fractal Figure representing the *nsTextHelper.cpp* product. The first rectangle rendered is the one corresponding to the author having the highest number of commits,

Author	Commits
warren@netscape.com	5
gerv@gerv.net	2
dcone@netscape.com	2
dbaron@fas.harvard.edu	1
cltbld@netscape.com	1
pierre@netscape.com	1
dmose@mozilla.org	1
jaggernaut@netscape.com	1
8 developers	14 commits

Table 1. Development distribution on the CVS product *nsTextHelper.cpp*.

namely `warren@netscape.com`. The area of the rectangle is $5/14$ of the total area since the number of commits performed by warren@netscape.com is 5 while the total number is 14. All the other rectangles are rendered in the same way, changing the direction of the longer side each time.

Treemaps. Fractal Figures have some resemblance to treemaps [8]. The main difference is the underlying data: treemaps are targeted at hierarchically organized data, while in our case it is raw, sorted, numerical data without any structure. Moreover, as we see later on, Fractal Figures allow for *gestalt* impressions to categorize them, while our experiments with treemaps did not yield the same effective results.

Layout Algorithm. Why do we swap the long side and the short side of the rectangles each time, instead of rendering always in the same direction? Our current layout algorithm makes the rectangles visible even when there are many, while experiments with rendering in the same direction (*i.e.*, splitting the whole rectangle always horizontally) does not scale. We claim that Fractal Figures are scalable, but what happens when there are hundreds of authors? In such a case, the Fractal Figure will only convey that the development is fragmented among many authors.

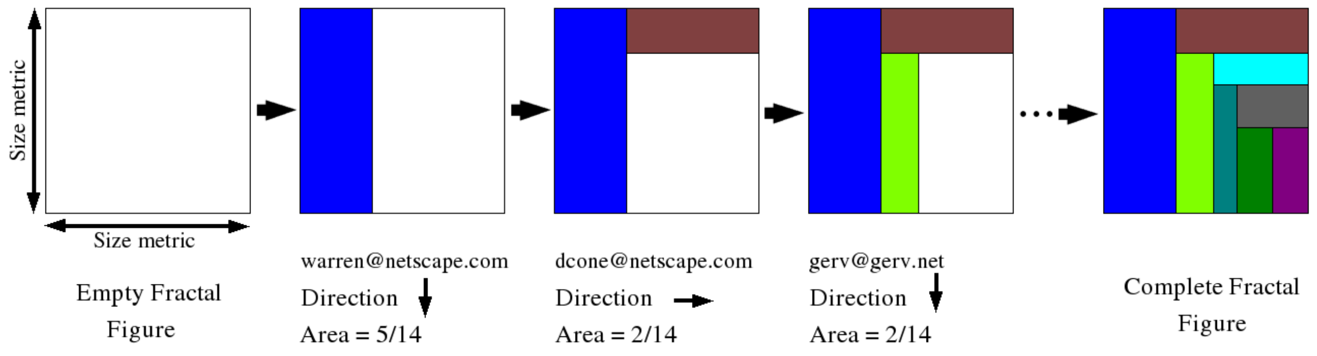

Figure 2. The structural principles of a Fractal Figure.

4. Fractal Figures in Action

Fractal Figures allow us to reason about the development effort of entities from the authors point of view. We can compare entities looking at their figures and classify them according to development models represented by well defined patterns.

4.1. Classifying CVS products with Fractal Figures

We distinguish 4 major development patterns, associated with Fractal Figures similar to those shown in Figure 3: Only one developer (a), only few developers and balanced effort (b), many developers but unbalanced effort—one performed half of the work, all the others performed the second half together—(c), and many developers performing roughly the same amount of work (d).

4.2. Using Fractal Figures in Polymetric Views

An efficient way to exploit the expressive power of Fractal Figures is in combination with polymetric views [9] to represent a set of entities, *i.e.*, products, directories or modules.

Examples of applicable software metrics in a CVS context are: Number of revisions, number of products (for directories and modules), number of bugs[1], number of authors, *etc.*

The metrics can also be aggregated for higher-level entities such as directories (*e.g.*, the number of revisions of a directory is the sum of the numbers of revisions of all contained CVS products), thus also allowing a categorization of those higher-level entities in terms of development effort and distribution.

1 The bugs information is not included in the CVS repository. See [3, 4] for how to retrieve such data.

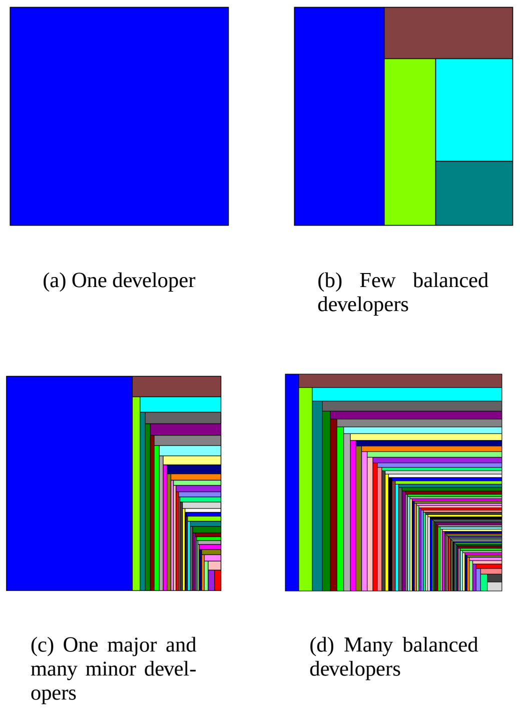
(a) One developer
(b) Few balanced developers
(c) One major and many minor developers
(d) Many balanced developers

Figure 3. Development patterns based on the *gestalt* of Fractal Figures.

4.3. Examples from Mozilla

In the following, we illustrate some example polymetric views using Fractal Figures. All the views are based on the Mozilla project. Due to space limitations we cannot present an in-depth analysis. For a complete analysis of the whole system, we refer the reader to [3].

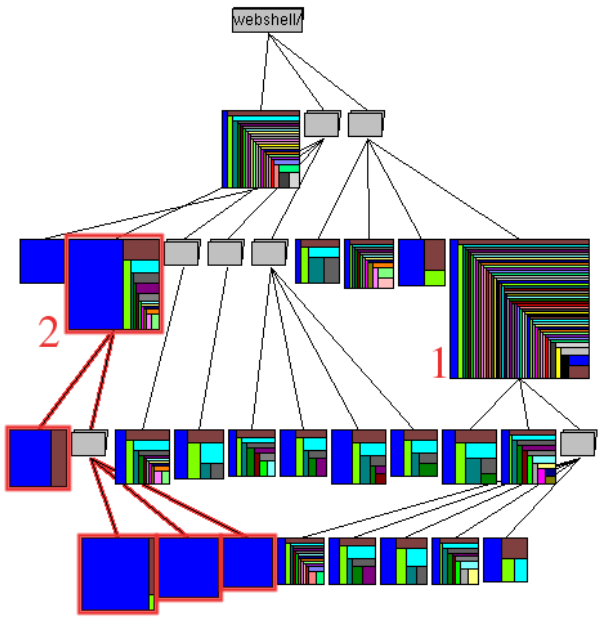

Figure 4. The webshell hierarchy of Mozilla.

Example 1 – Mozilla Webshell

Figure 4 shows the webshell hierarchy of Mozilla. The Fractal Figures represent directories including at least one product, while grey figures are associated with container directories, *i.e.*, directories containing only subdirectories. As size metric we use the number of products.

The figure marked as 1 represents the largest directory, *i.e.*, the directory containing the highest number of products. It exhibits a *many balanced developers* development pattern. The set of figures highlighted in red (marked as 2) is characterized by: (1) they belong to the same hierarchical structure; (2) they exhibit a *one developer, or mainly one developer* pattern; (3) some of them contain a great amount of products. These directories contain a large set of logically coupled products (because of the hierarchical structure) managed by only a few authors.

Example 2 - Mozilla XPInstall and LibEditor

Figure 5 and 6 show the xpinstall/src and editor/libeditor/html directories of Mozilla. Fractal Figures represent products and as size metric we use the number of revisions. The Fractal Figures representing the high level entities, *i.e.*, the directories, are also depicted.

The directory xpinstall/src contains figures having both similar sizes and similar patterns; a *balanced directory*, *i.e.*, all the products equally contribute to the directory development pattern (see Figure 5).

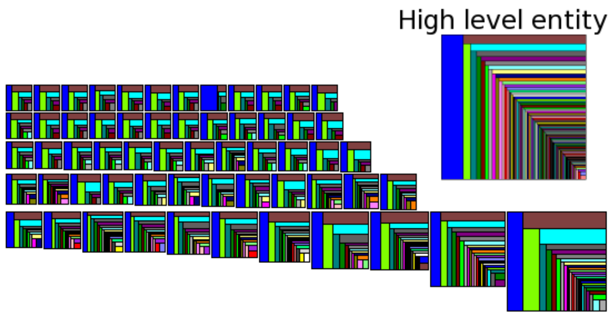

Figure 5. The xpinstall/src directory of Mozilla.

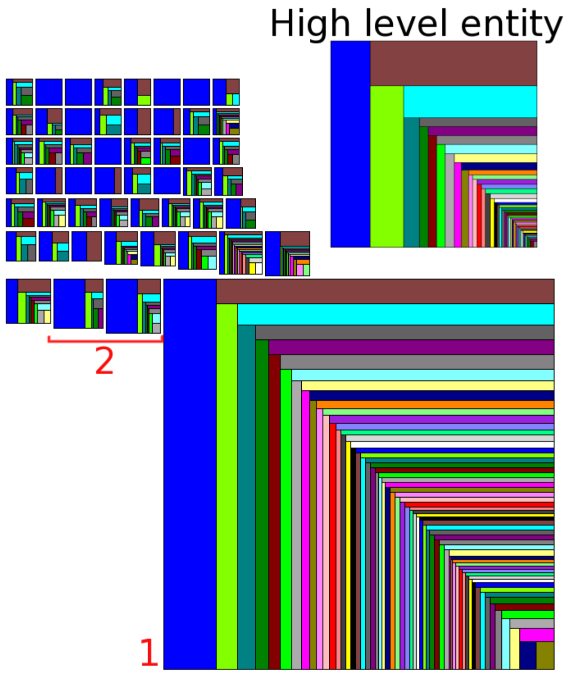

Figure 6. The editor/libeditor/html directory of Mozilla.

The directory editor/libeditor/html contains a huge figure (marked as 1), with respect to the average figure size. The development pattern of the product represented by this figure is reflected in the directory (*many balanced developers*). The products marked as 2 are also interesting. They are likely to be logically coupled because: (1) they have the same development patterns and a similar appearance; (2) their development patterns are different from that characterizing the directory.

5. Development patterns and the Fractal Value

In Section 4 we have seen how Fractal Figures can be enriched by mapping a metric measurement on their size. This feature not only increases the amount of information encapsulated in the figure, but also gives us the possibility to study the relationship between the development pattern (*i.e.*, the Fractal Figure appearance in terms of fragment numbers and layout) and the size.

Fractal Figures give us only a qualitative impression of the development pattern, while we need a quantitative measurement to postulate empirical laws. This measurement is given by the *Fractal Value*.

5.1. The Fractal Value

The Fractal Value measures how "fractalized" a Fractal Figure is, that is how much the work spent on the corresponding entity is distributed among different developers. It is formally defined as:

$$\text{Fractal Value} = 1 - \sum_{a_i \in A} \left(\frac{nc(a_i)}{NC} \right)^2, \quad (1)$$

$$NC = \sum_{a_i \in A} nc(a_i) \quad (2)$$

where $A = \{a_1, a_2, \ldots, a_n\}$ is the set of authors and $nc(a_i)$ is the number of commits performed by the author a_i.

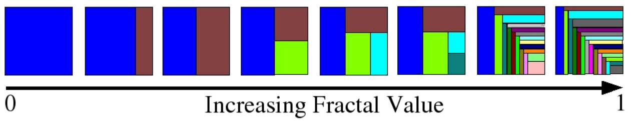

Figure 7. Visualizing the Fractal Value metric.

Looking at equation 1, we notice that:

- Since the square equation is sub-linear between 0 and 1, the smaller the quantity $\frac{nc(a_i)}{NC}$ is (always less than 1), the more it is reduced by the square power. Therefore, the smaller a rectangle is, the lesser its negative contribution to the Fractal Value is.
- The Fractal Value ranges from 0 to 1 (not reachable). It is 0 for entities developed by one author only, while it tends to 1 for entities developed by a large number of authors (see Figure 7). Table 2 shows the Fractal Values of the Fractal Figures of Figure 3.

Development Pattern	Fractal Value
One developer	0
One major and many minor developers	0.75
Few balanced developers	0.85
Many balanced developers	0.998

Table 2. Fractal Values of the four typical development patterns.

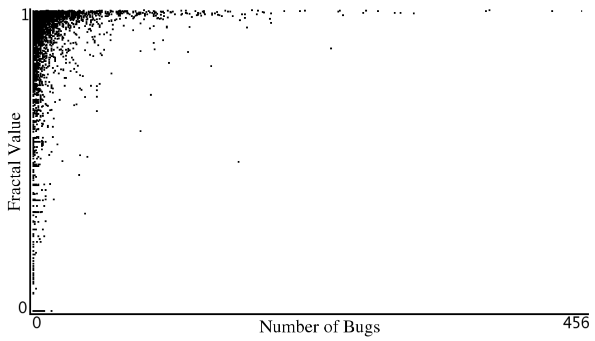

Figure 8. The relation between Fractal Value and number of bugs for Mozilla.

5.2. Correlation Views

Using the Fractal Value we can verify whether a relation between the development pattern and another software metric holds. In this case we choose to verify whether the number of authors is correlated with the number of bugs affecting a CVS product. We do that by representing products with fixed size squares and by mapping the Fractal Value and the considered metric (number of bugs) on their positions.

The resulting view is shown in Figure 8: It allows us to postulate the following empirical observation: "The more the development of a product is distributed among different authors, the greater is the number of bugs affecting it".

As a result, our Fractal Figures can be effectively combined with information about bugs per file to reason about the effectiveness of certain development patterns. For an analyst or a project manager, this relationship can be quite valuable since these visualizations allow him/her to reassess the current formation of the development teams. A right-sizing activity can almost immediately follow from the visualizations.

6. Related Work

Ball and Eick [1] concentrated on visualization of different aspects related to code-level such as code version history, difference between releases, static properties of code, code profiling and execution hot spots, and program slices. The basic concepts used in the visualization of the above aspects are colors and pixel representations of source code lines. In [7] Gall *et al.* presented an approach to use color and 3D to visualize the evolution history of large software systems. Colors were primarily used to highlight main events of the system evolution and reveal unstable areas of the system. Jazayeri analyzed the stability of the architecture [6] by using colors to depict the changes. Taylor and Munro [14] visualized CVS data with a technique called revision towers. This technique uses color bars of varying thickness and height to represent the current size, changes and authors of a piece of code. These bars are animated over time to show the development of the software repository. Collberg *et al.* [2] focus on the visualization of the evolution of software using a temporal graph model. They do not give any representation of the dimension of the effort. Rysselberghe and Demeyer [16] used a simple visualization based on information in version control systems (CVS) to provide an overview of the evolution of systems. Their visualization technique consists in a matrix where the columns represent files ordered by names and lines represent the time. In [12] Pinzger *et al.* proposed a visualization technique to study the evolution of large software systems. The approach provides integrated condensed graphical views on source code and release history data of many releases.

7. Conclusion

Open-source projects have many contributing developers. We have investigated the question, whether and by what means we can efficiently convey *how* much development effort particular files have accumulated over time and *what* developer contributed to which extent.

For that, we have presented different visualizations, stemming from a TIMELINE VIEW of CVS products and resulting in Fractual Figures of files. These visualizations exhibit development efforts and indicate development patterns such as the four we discovered: *one developer, only few developers and balanced effort, many developers but unbalanced effort, and many developers and balanced effort*.

Our Fractal Figures can be effectively combined with information about bugs per file to reason about the effectiveness of certain development patterns.

The resulting Fractal Figures allows us to postulate the following empirical observation: ”The more the development of a product is distributed among different authors, the greater is the number of bugs affecting it”.

For an analyst or a project manager, this relationship can be quite valuable since these visualizations allow him/her to re-assess the current formation of the development teams. A right-sizing activity can almost immediately follow from the visualizations.

References

[1] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, pages 33–43, 1996.

[2] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 77–86. ACM Press, 2003.

[3] M. D’Ambros. Software archaeology - reconstructing the evolution of software systems. Master thesis, Politecnico di Milano, Apr. 2005.

[4] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, Sept. 2003.

[5] D. Grune. Concurrent Versions system, a method for independent cooperation. Technical report, Vrije Universiteit, Amsterdam, Netherlands, 1986.

[6] M. Jazayeri. On architectural stability and evolution. In *Reliable Software Technlogies-Ada-Europe 2002*, pages 13–23. Springer Verlag, 2002.

[7] M. Jazayeri, H. Gall, and C. Riva. Visualizing software release histories: The use of color and third dimension. In *ICSM ’99 Proceedings (International Conference on Software Maintenance)*, pages 99–108. IEEE Computer Society, 1999.

[8] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc. of Visualization’91*, pages 284–291, San Diego, CA, 1991.

[9] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, Sept. 2003.

[10] M. M. Lehman and L. Belady. *Program Evolution — Processes of Software Change*. London Academic Press, 1985.

[11] Mozilla home page. http://www.mozilla.org/.

[12] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proceedings of SoftVis 2005, ACM Symposium on Software Visualization*, pages 67–76, St. Louis, Missouri, 2005.

[13] M. J. Rochkind. The Source Code Control System. *Transactions on Software Engineering*, 1(4):364–370, 1975.

[14] C. M. B. Taylor and M. Munro. Revision towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, pages 43–50. IEEE Computer Society, 2002.

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

[16] F. Van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of The 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, 2004. to appear.