JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 29, 17-34 (2013)

How Evolutionary Visual Software Analytics Supports Knowledge Discovery*

ANTONIO GONZÁLEZ-TORRES, FRANCISCO J. GARCÍA-PEÑALVO
AND ROBERTO THERÓN
Department of Computer Science
Faculty of Science
University of Salamanca
Plaza de los Caídos, 37008 Salamanca, Spain

Evolutionary visual software analytics is a specialization of visual analytics. It is aimed at supporting software maintenance processes by aiding the understanding and comprehension of software evolution with the active participation of users. Therefore, it deals with the analysis of software projects that have been under development and maintenance for several years and which are usually formed by thousands of software artifacts, which are also associated to logs from communications, defect-tracking and software configuration management systems. Accordingly, evolutionary visual software analytics aims to assist software developers and software project managers by means of an integral approach that takes into account knowledge extraction techniques as well as visual representations that make use of interaction techniques and linked views. Consequently, this paper discusses the implementation of an architecture based on the evolutionary visual software analytics process and how it supports knowledge discovery during software maintenance tasks.

Keywords: evolutionary visual software analytics, visual software analytics, visual analytics, human-computer interaction, software visualization, software evolution visualization

1. INTRODUCTION

Visual analytics is a process that iteratively collects information, preprocesses data, carries out statistical analysis [1], performs data mining, and uses machine learning [2], knowledge representation [3], user interaction [4], visual representations [5-7], human cognition [8], perception, exploration and the human abilities for decision making [9, 10].

It has been applied thoroughly to problems as diverse as avian flu [11], paleoceanographic conditions [12], organization analysis [13], decision making [14, 15], temporal patterns [16, 17], social networks [18], security analysis [19] and software systems [20, 21]. Therefore, one can say that knowledge discovery is an intrinsic property of visual analytics, as it is aimed at supporting analysts in gaining insight from large multivariate datasets [22].

The visual analytics tasks are named after components of the process and are the following: the knowledge extraction engine, the data transformation engine for visualization models, the visual knowledge explorer and the user [23]. Such process begins with

Received May 31, 2011; accepted March 31, 2012.
Communicated by Francisco J. García-Peñalvo, Ricardo Colomo-Palacios and Jane Yung-Jen Hsu.
* This work was supported by Spanish Government project TIN2010-21695-C02-01, by the Castile and Lion Regional Government through GR47, by the Ministerio de Ciencia e Innovación of Spain under project FI2-010-16234 and also by the Ministry of Science and Technology (MICIT) of Costa Rica.

the retrieval of data from heterogeneous data sources which then is processed by the knowledge extraction engine using one or more techniques for discovering, representing and managing knowledge. Knowledge extraction techniques include information retrieval, knowledge representation and management, data mining, statistical analysis, machine learning, compression and filtering and semantic based approaches.

Although the aim of applying one or more knowledge extraction techniques to large data sets is to support analysts by producing condensed data sets of relevant facts, usually those resulting data sets are still large and complex. Thus, the visual knowledge explorer is complementary to the knowledge extraction engine. It is comprised of techniques for presenting information and aiding the comprehension and understanding of facts through relationships. To this end it uses linked views, interaction and usability techniques to provide a tool to users for exploring and applying human cognition, perception, sensemaking, analytical reasoning and critical thinking for decision making.

This paper is aimed at evolutionary visual software analytics, a specialization of visual analytics that is concerned with knowledge discovery for supporting software maintenance processes. Therefore, evolutionary visual software analytics addresses the analysis of large software projects whose life cycle usually expands through several years, generates thousands or even millions of lines of source code (LOC) [24], hundreds of software components and thousands of revisions [25]. Furthermore, it also deals with the relationships among software items in the form of inheritance, interface implementation, coupling and cohesion. In addition, source code details such as variables, constants, programming structures, methods and relationships among those elements must be considered. Besides, logs, communication systems, defect-tracking systems and SCM tools keep records with dates, comments, changes made to software projects, associated users and programmers [26].

Accordingly, section 2 describes and discusses the visual software analytics and evolutionary visual software analytics processes; section 3 explains an architecture for evolutionary visual software analytics; section 4 expounds some visualization designs for socio-technical analysis within software maintenance; and section 5 states the main conclusions.

2. BACKGROUND

Understanding the evolution of a software project is a complex process that requires the automatic analysis of the project evolution, the visual representation of such analysis and the active participation of users in the knowledge discovery process by interacting with the visual representations. The aforementioned analysis takes into consideration the evaluation of individual revisions and the comparison of the output produced by such evaluation for a given number of revisions or all the existing revisions associated to the project. Thereafter, the output of the analysis is visually represented and appropriate interaction techniques are added to the visual representations. The aim is to involve the active participation of users in the knowledge discovery and comprehension processes of relevant facts regarding the evolution of the software project.

Consequently, the next sections focus on the discussion of the evolutionary visual software analytics process and software evolution visualization.

2.1 Evolutionary Visual Software Analytics

The evolutionary visual software analytics process is a specialization of visual software analytics, which in turn is based on visual analytics. Therefore, the evolutionary visual software analytics process described in Fig. 1 is shared by visual software analytics as well as by visual analytics. So, the generic process starts with the retrieval of data from heterogeneous data sources (reads data, arrow 1). Then, the retrieved data is processed by the knowledge extraction engine using one or more knowledge extraction techniques for discovering, representing and managing knowledge (produces facts, arrow 2). In turn, the output of the knowledge extraction engine is stored in the facts database. Then, the facts analyzer engine reads facts from the software evolution facts database (reads facts, arrow 3) and process them appropriately. Following, the data transformation engine for visualization models takes the analysis results from the facts analyzer engine (first analysis results, arrow 4) and creates the appropriate data structures required by the visual knowledge explorer to display the most important results (shows the important outcome, arrow 5). Next, using a combination of techniques the visual knowledge explorer provides a tool to users which allows knowledge exploration and discovery (zoom, filter, interact, arrow 6).

This process is iterative and requires additional details as the user interacts, explores and discovers knowledge through the creation of associations and relationships among visual elements in the visual knowledge explorer (request of details on demand, see arrow 7). Thus, the visual knowledge explorer automatically requests additional information for providing more details to users and if the requested data is unavailable from the persistent data structures created by the data transformation engine, the facts analyzer engine performs further analysis tasks (further analysis, arrow 8). Consequently, those results are passed to the data transformation engine and added to the persistent data structures (further analysis results, arrow 4). Finally, the corresponding details are visually represented in the visual knowledge explorer (details on demand, arrow 5) and the user continues working on the knowledge discovery process until he/she decides to stop once the proposed goals have been reached.

Furthermore, evolutionary visual software analytics and visual software analytics also share their aim of supporting the study and analysis of the dynamics of software systems for aiding software maintenance processes [26]. Although, the main difference between them is that evolutionary visual software analytics looks to support the comprehension of the overall software project evolution while visual software analytics supports the comprehension of the current state of the software project, without taking into account its past. Therefore, visual software analytics only takes into consideration a revision of the software project whereas evolutionary visual software analytics considers all the revisions of the software project. A practical analogy for explaining such difference between evolutionary visual software analytics and visual software analytics is that the first could be seen as a movie while the latter would be a movie frame.

Therefore, evolutionary visual software analytics requires to compare the results carried out on individual revisions. The techniques used by the knowledge extraction engine, as well as the software facts that are extracted, and the software visualization techniques, vary according to the aims of the research or tool design. However, the use of interaction techniques and how visualizations are linked plays a relevant role in the

knowledge discovery process. Programmers and project managers make use of human cognition, perception, sense-making, analytical reasoning and critical thinking to find relevant facts and relationships to understand the software project by means of human interaction. Similarly to visual analytics, the evolutionary visual software analytics process stops when users have completed the knowledge discovery task that has been carried out, or no satisfactory answer has been found.

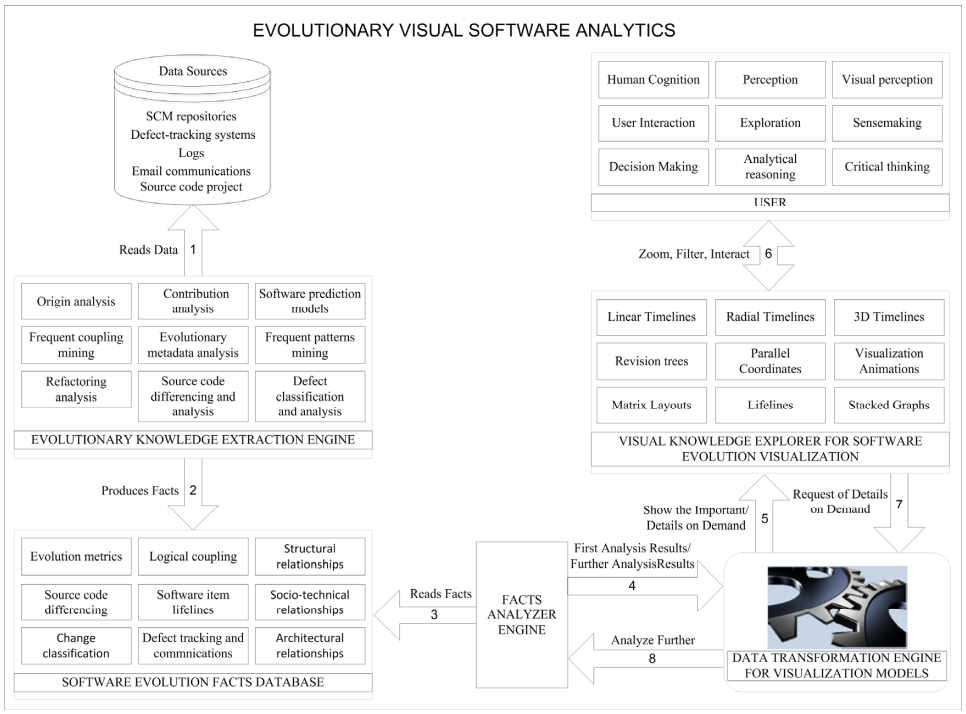


Fig. 1. The evolutionary visual software analytics process.

The complexity of evolutionary visual software analytics is higher than in visual software analytics because of the additional dimension added by the analysis of evolution and the representation of time. Therefore, the techniques used by the knowledge extraction engine, the facts stored by the software evolution facts database and the visual knowledge explorer for software evolution visualization take into account the complete evolution of the software project or a given period of time.

The aim of the knowledge extraction engine is to produce software facts, such as evolution metrics, logical coupling details, structural relationships, source code differencing, software item lifelines, socio-technical relationships, defect tracking and communications, and architectural relationships. For this purpose, it makes use of several analysis techniques: origin analysis, contribution analysis, software prediction models, frequent coupling mining, evolutionary metadata analysis, frequent patterns mining, refactoring analysis, source code differencing, and defect classification and analysis. Be-

sides, the visual knowledge explorer for software visualization makes use of several visualization techniques for representing software facts. Among the techniques used are polymetric views, matrices, dependency graphs, software cities, UML-based diagrams, call graphs, clustering and heatmaps techniques, algorithm animation, and socio-technical views.

2.2 Software Evolution Visualization

In software evolution, change events can be seen at different scales, or granularities [27]. Several visualizations address the problem of getting insight in time series [28]. However, this is not enough for software-related problems, as stakeholders need to correlate time events with data changes, such as structural or metric changes [24, 29]. As such, researchers have investigated ways to combine time events with data changes in timelines, *i.e.*, the representation or exhibition of key events within a particular historical period, often consisting of illustrative visual material accompanied by written commentary, arranged chronologically.

In information visualization, several visualizations focus on correlating time events with structure or metrics (values). For example, SemTime uses a set of stacked timelines for the same or different time ranges, decorated with arrows to show relationships between data elements in the timelines [30]. Continuum combines the use of a timeline with a scalable histogram overview and allows the navigation through a complete hierarchical data set [31]. Treevolution introduces a tree ring metaphor to represent hierarchical time-based structures to browse and discover relationships in the history of computer languages [32]. Spiral Graph employs a spiral metaphor to show a timeline to visualize large time data sets, compare values along time, and detect periodic behaviors and trends [33]. The Spiral Semantic Timeline [34] takes advantage of a radial layout and accumulates activities in each time period. This visualization uses semantic zoom to allow pattern discovery by years, months, weeks, days and hours.

The use of timelines in the visualization of software evolution is often linked to the representation of software project hierarchies with the use of treemaps, graphs (using different layouts), and cone trees. Two related approaches to our work that have been applied to visualize software project hierarchies are [35] and [36]. The evolution visualizations of Pinzger *et al.* [29] and Voinea *et al.* [37], show the evolution of metrics over thousands of software artifacts and hundreds of changes, apply to a higher level of granularity.

Without detailing further due to space limitations, however, one can see several unsolved, yet general, challenges in software evolution visualization. First, one may want to visualize data at different time scales (years, months, days, hours) and also correlate the different scales. Secondly, visualizing software structural relationships is still difficult, although much work has been done in graph animation and evolution. Last but not least, it is crucial to intuitively combine different types of visualizations when addressing a real-world task, as outlined by visual analytics research [38]. While not claiming to fully address such goals, our work here presents several steps in designing such a solution for software evolution analysis purposes.

3. ARCHITECTURE DESIGN FOR THE EVOLUTIONARY VISUAL SOFTWARE ANALYTICS PROCESS

This section describes and discusses an architecture based on the evolutionary visual software analytics process (see Fig. 2). The goal of such architecture is to provide a reference design for novel researchers and tool designers interested in the evolutionary visual software analytics process. Its implementation has been carried out in Java and tested with several open source software projects such as jEdit, JabRef, Jmol, and JFreechart.

The knowledge extraction engine on an evolutionary basis supports the addition of modules for backing up new software configuration management systems (SCM) and allows configuring connections to several different projects and software repositories for extracting evolutionary details and carrying out software evolution analysis. Its components are: the monitor of new revisions, the source code extractor, the source code parser and the metadata and software evolution analysis engine.

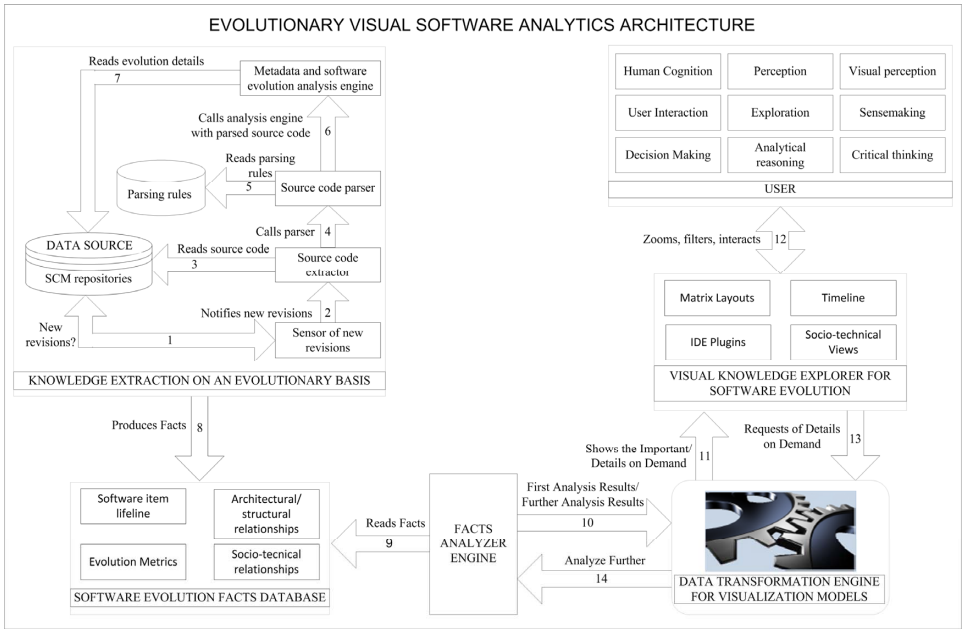


Fig. 2. Architecture for the evolutionary visual software analytics process.

The monitor of new revisions is a process that continuously monitors the addition of new revisions to software projects and notifies about new revisions to the source code extractor. Then, the source code extractor starts extracting source code from the software repository and invokes the source code parser for collecting details about the software project structure, the hierarchy of classes, the coupling relationships and the source code and metrics raw data for calculating metrics for classes and methods. Finally, the metadata and software evolution analysis engine takes the outcomes produced by the source code parser and queries additional details from the software repository. Then, it applies an exhaustive software evolution analysis and stores the results in the software evolution

facts database.

The knowledge extraction engine and the software evolution facts database are server side components with a graphical interface. The facts analyzer engine and the data transformation engine are a middleware processes that are configured using a graphical interface and are executed automatically when new additions to the software evolution facts database are detected.

Because our interest in analyzing the correlation between the contributions made by programmers and the structural changes in software project evolution, the software evolution facts that have been considered for the implementation of the architecture are the software item lifelines, the evolution metrics, the socio-technical relationships and some architectural/structural relationships such as inheritance, interface implementation, and the correlation of structural data with metrics. Therefore, the data transformation engine for visualization models transforms the software evolution facts into the appropriate data structures that are used by software evolution visualizations.

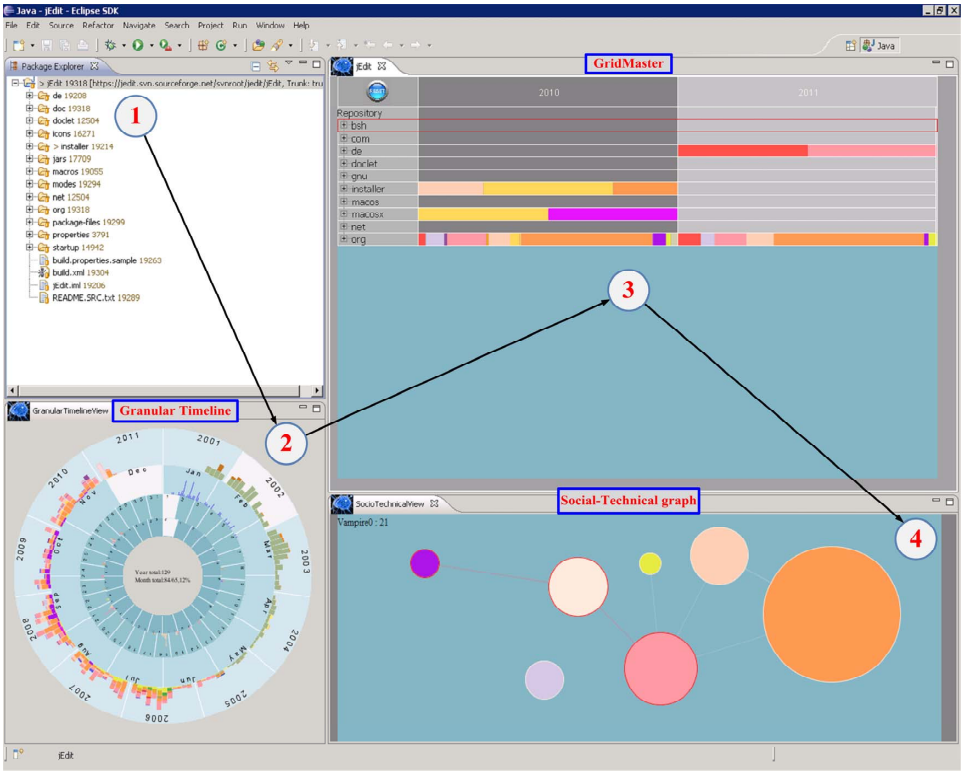


Fig. 3. Main view of the visual knowledge explorer.

4. VISUAL KNOWLEDGE EXPLORER FOR SOFTWARE EVOLUTION

The visual knowledge explorer for software evolution has been developed in Java as an Eclipse plugin. It makes use of linked views (organized by an overview + details

approach), interaction techniques and a color code for representing programmer's contributions. Fig. 3 shows the views that it includes: the Granular Timeline (displayed at the left bottom corner), the Grid Master (located at the right top panel) and a social-technical graph (depicted at the right bottom panel). Accordingly, the Granular Timeline provides an overview of the project activities and the Gridmaster, as well as the socio-technical graph, provide specific details regarding the correlation of project structure, associations and time. Thus, after the visualizations are launched from the contextual menu of the Eclipse's Package Explorer view, the knowledge discovery workflow (as indicated by numbers and arrows in Fig. 3) begins with the analysis of the programmers' contribution patterns in the Granular Timeline, and the selection, for further analysis in the Gridmaster, of one or more time units from this radial view. Then, the user can select one of the displayed time units in the Gridmaster for presenting the associated socio-technical relationships in the corresponding view.

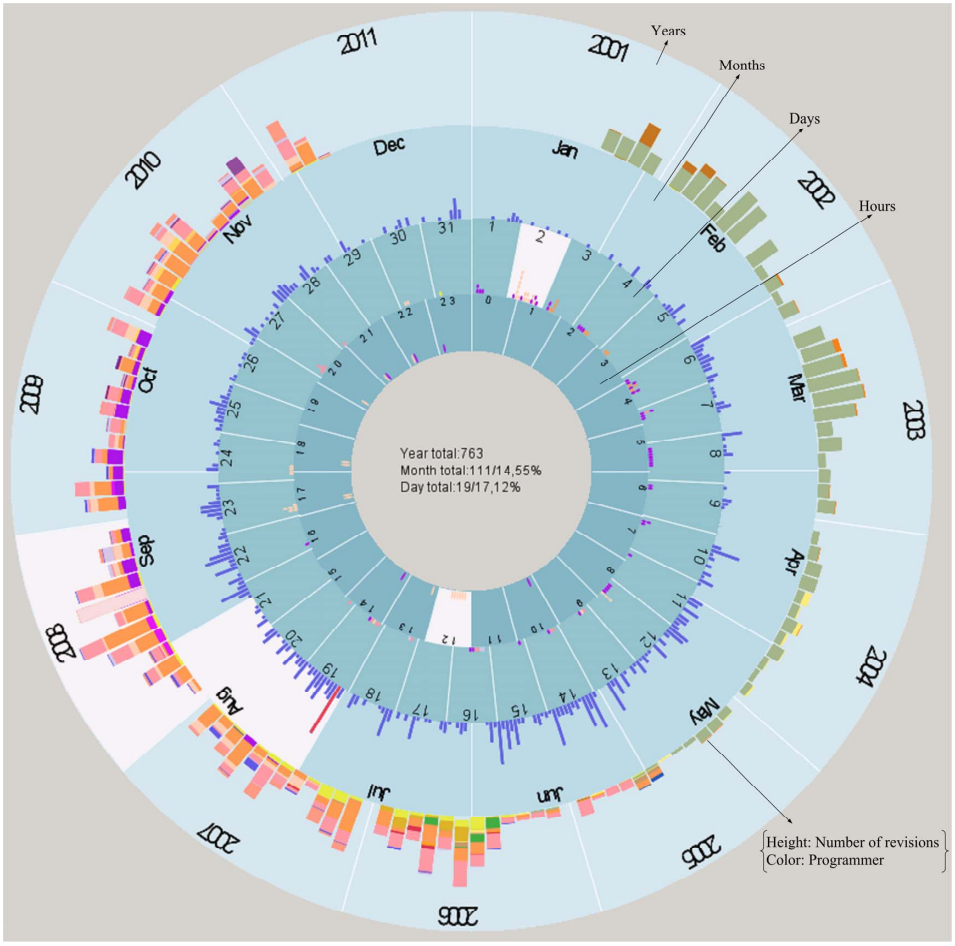


Fig. 4. Granular timeline showing statistics for revisions committed for the jEdit source open software project during 10 years.

Accordingly, the next sections explain the decisions taken for the visual representations and how they contribute to knowledge discovery in the context of software evolution.

4.1 Granular Timeline

The granular timeline uses a modified circular ring chart layout, to show an entire overview of the time dimension of a project (Fig. 4). Concentric rings demonstrate the different time scales that record change events, from coarse (years, outer ring) to fine grain (hours or finer, innermost ring). A related layout was used by Holten *et al.* to depict software project hierarchies [28]. This type of layout compactly presents large quantities of data and provides an overview + detail view.

As opposed to most applications using circular ring charts, such as [35], the space within each chart cell is used to embed different types of visualizations for representing the number of revisions at that cell's level of detail. First, the 'years' ring cells insert bar charts that show the number of revisions for each month for each year.

The 'months' ring embeds a height plot chart presenting the number of committed revisions per day-of-month. Next, the 'days' ring shows revisions within each day, so this ring has 30 or 31 cells. In each cell, a matrix dot plot is drawn in polar (ρ, ϕ) coordinates, where ρ maps the creation hour of a revision. Finally, the innermost 'hours' ring shows revisions by hour, so it has 24 cells.

Additionally, statistics are displayed in the center of the visualization as time units are selected. Note that time unit selections start in the outer ring that corresponds to 'years' and follows the sequence months → days → hours, as highlighted in Fig. 4 (2008 → August, 2008 → August 2nd, 2008 → August 2nd, 2008 at 12).

4.2 Gridmaster

This visualization is based on a matrix-like representation for being a simple structure widely known by programmers and because it allows the easy establishment of relationships among multivariate elements. The cells of the matrix representation, with the packages and software items in the rows and the time units in the columns, are used for the correlation of details associated to the corresponding package or software item. Basically, the evolution details that are correlated with the project structure are programmer contributions, the creation of software items, the addition or removal of inheritance or interface implementation relationships, and software item metrics.

Colors are associated to programmers and the area associated to each one depends on the number of contributions, in terms of committed revisions, as well as on the representation that has been chosen from the contextual menu: relative or absolute. In this context, $\varepsilon = \Delta/\eta$ is the area assigned to any time unit in the visual representation, where Δ is the dimension of the graphic area in pixels and η is the number of time units (years, months or days) in the visual representation. Then, for the relative representation, $\alpha = \varepsilon/\tau$ is the area assigned to a programmer contribution, where τ is the number of contributions for the time unit with more contributions. Therefore, the area assigned to a given programmer for a given time unit is $\Lambda = \alpha * \beta$, where β is the number of contributions made by the programmer during that time unit. For the absolute representation, the area as-

signed to a programmer contribution is different for each time unit. So, $\gamma = \varepsilon/\omega$, where ω is the number of contributions for the time unit under consideration.

The visual representation of the project structure is made up of all the packages and software items that have been added to the project during its evolution. This allows correlating all software items involved in the evolution process with programmers, metrics, and architectural relationships such as inheritance and interface implementation relationships. On the left side, Fig. 5 shows the current project structure of jEdit in the Eclipse workbench and depicts the project and its corresponding structure inside the visualization, including packages and software items that are no longer part of the current project version. Such design feature allows representing the lifeline of software items and packages using an intuitive approach, when the absolute representation is chosen as shown in Fig. 5. The details view on the right side of the visualization presents that the performed activities on the package "bsh" were carried out between 2001 and 2006. Moreover, that package currently is not part of the latest version of the project, which is corroborated when reviewing the project structure on the Eclipse workbench. So, the lifeline of a package or software item is determined by the representation of programmer's activities in the matrix view that is depicted by the use of colors.

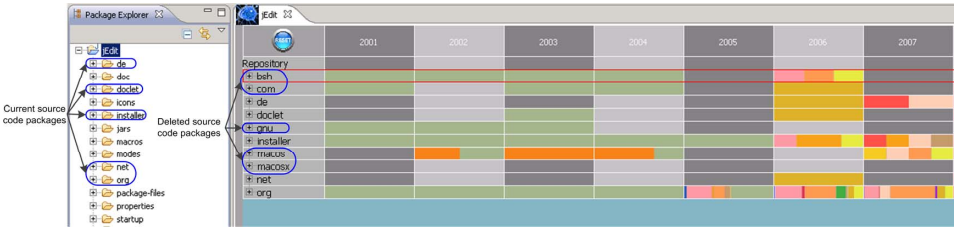


Fig. 5. Absolute representation of programmer's contributions showing project structure and lifelines.

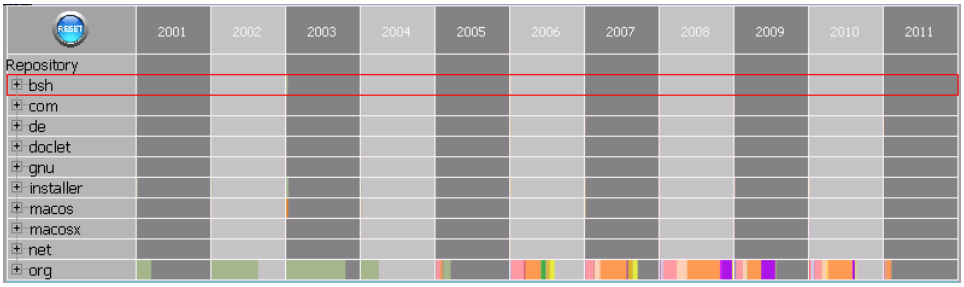


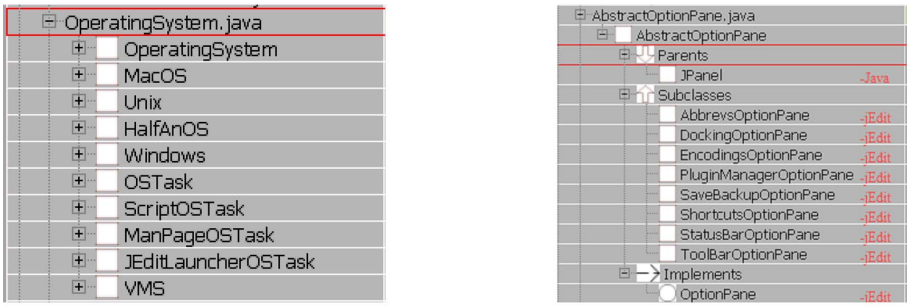
Fig. 6. Relative representation of programmers' contributions using 2008 as the reference year.

On the other hand, the relative representation allows comparing with precision the activity carried out in packages as well as the time period that concentrates on more activities, at the time that also provides visual information regarding programmer's contribution, as the area assigned to a contribution is the same for all time units in the visualization. Fig. 6 shows that the package with more activities is "org", the year in which more activities were carried out is 2008, and the programmer with more activity during the period 2001 to 2005 is the one represented by the user spestov (whose name is ob-

tained from the tooltip that is displayed when the mouse is moved over the colored areas).

The representation of the project structure in Gridmaster is composed by foldable nodes. Packages contain source code files and source code files contain software items, whereas software items include details about inheritance and interface implementation relationships, as illustrated by Figs. 7 (a) and (b). Complementary, Fig. 8 shows inheritance and interface implementation relationships are conveyed by a green oval, and its termination is represented by a red oval. In addition, the location of associated software items is explicitly indicated (Java, current project or external library). Software item metrics are represented using bar charts with the aim of highlighting changes. Similar to the representation of programmer contributions, metrics are represented using relative and absolute areas. Relative representation takes into consideration the highest metric value associated to the software item and the absolute representation takes into account the software item with the highest metric value for calculating the chart height.

Software item metrics are represented using bar charts for highlighting changes in metrics (see Fig. 9). Similar to the representation of programmer contributions, metrics



(a) Software items contained by a file. (b) Inheritance and interface implementation relationships.

Fig. 7. File contents and relationships.

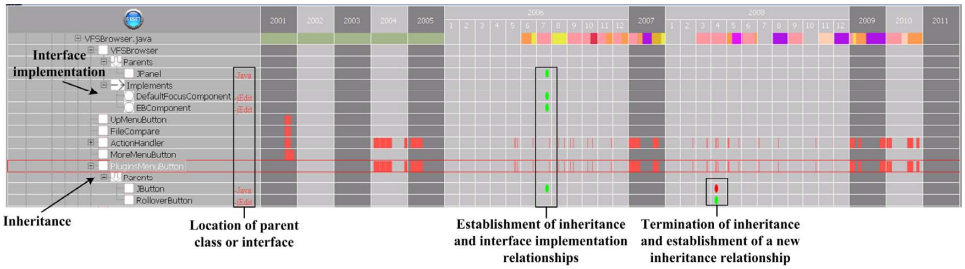


Fig. 8. Inheritance and interface implementation relationships.

Fig. 9. Representation of metrics.

are depicted using absolute and relative areas. The relative representation takes into consideration the software item with the highest metric value for calculating the chart height. On the contrary, the absolute representation only considers the highest metric value associated to the software item.

The interactions supported by Gridmaster include the possibility of zoom-in and zoom-out, the fisheye distortion, the reorder of project structure elements, and the capability of filtering out nodes from the structure. In addition, it supports year selection from the timeline for depicting data according to associated months. Moreover, the user has the possibility to choose how metrics and programmers are represented by selecting between relative or absolute value representations.

5. KNOWLEDGE DISCOVERY SCENARIO: SOCIO – TECHNICAL RELATIONSHIPS

The visual knowledge explorer components discussed above allows knowledge discovery for several software evolution facts such as the lifeline of software items, the evolution of inheritance and interface implementation relationships, metrics and socio-technical relationships. Accordingly, this section focuses on knowledge discovery from socio-technical relationships.

The Granular Timeline represents contributions using bar charts (Fig. 4) or a combination of bar charts and treemaps [6] (Fig. 10). The bar chart representation provides details about the number of contributions at each granularity level. However, the representation of data at the 'days' and 'hours' granularity levels does not take full advantage of the small graphic area available. Thus, treemaps were applied as they employ a space filling algorithm, which makes better use of space, and additionally allow to better highlight individual revisions.

Analyzing the contribution patterns is an intuitive task with the Granular Timeline. Fig. 11 shows the contributions made to jEdit during 11 years. The use of color is an important feature in this representation and the area covered in the graph by a given color is proportional to the number of contributions made by a given programmer. Accordingly, it is easy to realize, at first glance, that during the years 2001, 2002, 2003, and 2004 most of the contributions were made by the programmer represented by the light green color, which name is spestov (note that the names of programmers are obtained from a tooltip that appears when the mouse moves over the corresponding graphic area). However, in 2005 the contributions were made by 7 programmers (this is observed from the colors representing programmers), being spestov and ezust (dark pink) the ones with more contributions. Later, in 2006 the programmer with more contributions is ezust and the number of contributions made by kpouer (orange) has increased, and no contributions were made by the user spestov since May, 2005. Therefore, one can conclude that spestov left the project by that month. Thereafter, ezust and kpouer have shared the maintenance and development of the project until 2011, with a relevant number of contributions made by vanza (yellow) and shlomy (purple).

Contributions patterns could be observed in the Granular Timeline as well as in the GridMaster. However, the GridMaster correlates contributions with the project structure down to the file level, as it can be noted in Fig. 3. This allows project managers to deter-

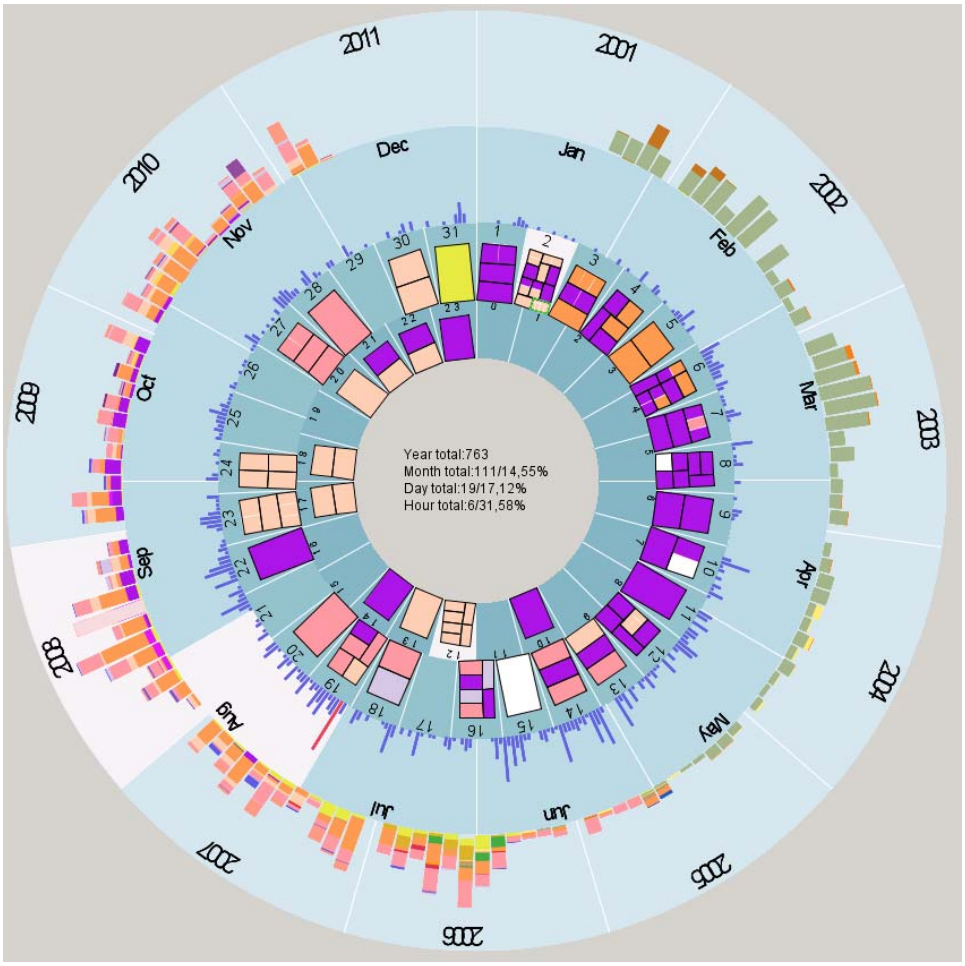


Fig. 10. Granular timeline using treemaps for highlighting contributions.

mine which software items have been changed by each programmer and assign programming tasks to programmers according to their previous experience.

The socio-technichal graph is a simple and powerful view that is displayed when a time unit, a year or month, is selected from the GridMaster. In this scenario, it is a complementary view for visualizing the relationship between programmers, based on the software items they have changed in common. The nodes of the graph represent programmers and their size conveys the number of contributions they have made (see Fig. 11). Edges connecting programmers depict relationships among these programmers based on changes to software items. The thickness of edges represents the number of common software items that the associated programmers have changed. Accordingly, kpouer is the programmer that has made more contributions in the selected year (2010), followed by ezust, k_satoda (light yellow), and daleanson (light gray). Additionally, kpouer has changed several software items in common with ezust, k_satoda, and daleanson. Thus,

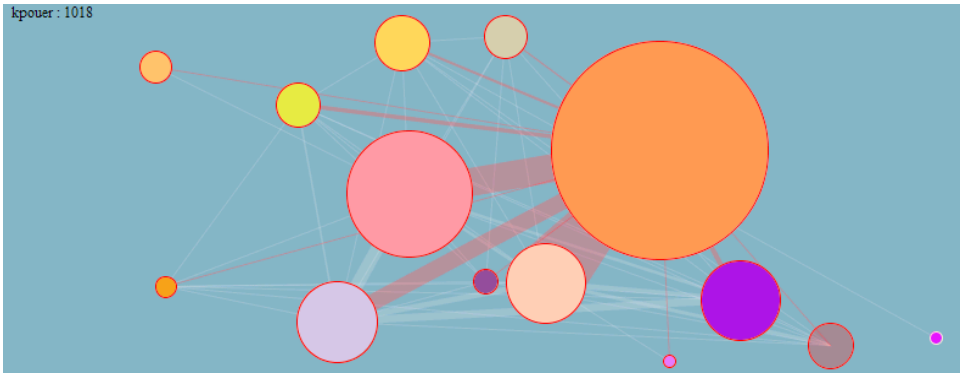


Fig. 11. Socio-technical graph showing the relationships between programmers based on the software items they have modified in common.

the relationship between kpouer and ezust is very strong due to the large number of software items they have changed in common. Therefore, this visualization is useful in scenarios where programmer's tasks need to be redistributed to an unexpected situation or organization change. Consequently, programming tasks performed by kpouer could be eventually taken over by ezust, k_satoda, and daleanson.

6. CONCLUSIONS

The main contribution of this paper is defining of the evolutionary visual software analytics process, due to its usefulness in the design of the tools and architectures that look for supporting software maintenance tasks. Accordingly, it has explained in detail the definition of such process and its main components. In addition, it has also discussed the design and implementation of an architecture based on the aforementioned process.

Consequently, the implementation of this architecture has allowed to test the feasibility of applying the evolutionary visual software analytics process to software maintenance tasks and determine if it could effectively contribute to knowledge discovery within software evolution. Hence, the use of the resulting tool during controlled tests at the lab has showed that evolutionary visual software analytics can effectively contribute to software maintenance tasks by providing information about structural project details, software metrics, and socio-technical relationships. Although it is important to highlight that the knowledge elements, which any tool based on this process could provide, can be related to any software evolution detail, according to the developer's interest and the specificity of the implementation. Thus, the evolutionary visual software analytics process, as well as its usefulness in knowledge discovery tasks, has been partially tested and a more rigorous validation with a group of users is pending for showing its usefulness in day to day tasks.

Undoubtedly, the application of information visualization, and specifically visual analytics, to any problem that involves data analysis, allows to grasp easily and quickly (within a few minutes) the details that otherwise remain hidden from users. Therefore, one of the strongest points of the evolutionary visual software analytics process in knowledge discovery tasks is the use of visualization techniques.

Finally, an additional contribution of this research is that it could be used as reference for guiding other researchers to define specific visual analytic processes for their own research areas (*i.e.* visual ontology analytics and evolutionary visual e-learning analytics).

REFERENCES

1. R. Peck, C. Olsen, and J. L. Devore, *Introduction to Statistics and Data Analysis*, 4th ed., Brooks/Cole, Independence, Cincinnati, 2011.
2. F. Harmelen, V. Lifschitz, and B. Porter, *Handbook of Knowledge Representation*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 2008.
3. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann, Oxford, United Kingdom, 2011.
4. Y. Rogers, H. Sharp and J. Preece, *Interaction Design: Beyond Human-Computer Interaction*, 3rd ed., John Wiley & Sons, Inc., Hoboken, New Jersey, 2011.
5. Y. K. Leung and M. D. Apperlley, "A review and taxonomy of distortion-oriented presentation techniques," *ACM Transactions on Computer-Human Interaction*, Vol. 1, 1994, pp. 126-160.
6. B. Johnson and B. Shneiderman, "Treemaps: a space-filling approach to the visualization of hierarchical information structures," in *Proceedings of the 2nd Conference on Visualization*, 1991, pp. 284-291.
7. G. G. Robertson, J. D. Mackinlay, and S. K. Card, "Cone trees: animated 3d visualizations of hierarchical information," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1991, pp. 189-194.
8. A. Drigas, L. Koukianakis, and Y. Papagerasimou, "Towards an ICT-based psychology: e-psychology," *Computers in Human Behavior*, Vol. 27, 2011, pp. 1416-1423.
9. D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler, "Challenges in visual data analysis," in *Proceedings of IEEE Conference on Information Visualization*, 2006, pp. 9-16.
10. X. Llorá, K. Sastry, F. Alías, D. E. Goldberg, and M. L. Welge, "Analyzing active interactive genetic algorithms using visual analytics," in *Proceedings of ACM Conference on Genetic and Evolutionary Computation*, 2006, pp. 1417-1418.
11. P. Proulx, S. Tandon, A. Bodnar, D. Schroh, R. Harper, and W. Wright, "Avian flu case study with space and geotime," in *Proceedings of IEEE Symposium on Visual Analytics Science And Technology*, 2006, pp. 27-34.
12. R. Theron, "Visual analytics of paleoceanographic conditions," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2006, pp. 19-26.
13. S. K. Card, S. Bongwon, B. A. Pendleton, and H. Jeffrey, "TimeTree: Exploring time changing hierarchies," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2006, pp. 3-10.
14. M. Migut, J. van Gemert, and M. Worring, "Interactive decision making using dissimilarity to visually represented prototypes," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2011, pp. 141-149.
15. A. Savikhin, R. Maciejewski, and D. Ebert, "Applied visual analytics for economic decision-making," in *Proceedings of IEEE Symposium on Visual Analytics Science*

and Technology, 2008, pp. 107-114.
16. C. Weaver, D. Fyfe, A. Robinson, D. Holdsworth, and D. Peuquet, "Visual analysis of historic hotel visitation patterns," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2006, pp. 35-42.
17. H. Ziegler, M. Jenny, T. Gruse, and D. Keim, "Visual market sector analysis for financial time series data," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2010, pp. 83-90.
18. A. Perer, I. Guy, E. Uziel, I. Ronen, and M. Jacovi, "Visual social network analytics for relationship discovery in the enterprise," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2011, pp. 71-79.
19. L. Harrison, W. Dou, A. Lu, W. Ribarsky, and X. Wang, "Guiding security analysis through visualization," in *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, 2011, pp. 317-318.
20. A. Gonzalez-Torres, R. Theron, A. Telea, and F. J. García-Peñalvo, "Combined visualization of structural and metric information for software evolution analysis," in *Proceedings of ACM Workshops on Principles of Software Evolution and Software Evolution Workshops*, 2009, pp. 25-30.
21. A. González-Torres, R. Therón, F. J. Garcia-Penalvo, M. Wermelinger, and Y. Yu, "Maleku: An evolutionary visual software analytics tool for providing insights into software evolution," in *Proceedings of IEEE International Conference on Software Maintenance*, 2011, pp. 594-597.
22. J. J. Thomas and K. A. Cook, *Illuminating the Path: Research and Development Agenda for Visual Analytics*, IEEE Press, Los Alamitos, CA, 2005.
23. A. González-Torres, F. J. García-Peñalvo, and R. Therón, "Human-computer interaction in evolutionary visual software analytics," *Computers in Human Behavior*, in Press, doi:10.1016/j.chb.2012.01.013
24. H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 19, 2007, pp. 77-131.
25. M. D'Ambros, H. C. Gall, M. Lanza, and M. Pinzger, "Analyzing software repositories to understand software evolution," *Software Evolution*, Springer, Berlin, Germany, 2008, pp. 37-67.
26. A. E. Hassan, "Mining software repositories to assist developers and support managers," Ph.D. Thesis, Department of Computer Sciences, Waterloo University, Canada, 2005.
27. U. D. Lago, A. Montanari, and G. Puppis, "On the equivalence of automaton-based representations of time granularities," in *Proceedings of the 14th IEEE International Symposium on Temporal Representation and Reasoning*, 2007, pp. 82-93.
28. H. Hochheiser and B. Shneiderman, "Dynamic query tools for time series data sets: timebox widgets for interactive exploration," *Journal of Information Visualization*, Vol. 3, 2004, pp. 1-18.
29. M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing multiple evolution metrics," in *Proceedings of ACM Symposium on Software Visualization*, 2005, pp. 67-75.
30. M. Jensen, "Visualizing complex semantic timelines," Technical Report NBTR2003-001, NewsBlip, 2003.

31. P. André, M. L. Wilson, A. Russell, D. A. Smith, A. Owens, and M. C. Schraefel, "Continuum: Designing timelines for hierarchies, relationships and scale," in *Proceedings of Symposium on User Interface Software and Technology*, 2007, pp. 101-110.
32. R. Therón, "Hierarchical-temporal data visualization using a ring tree metaphor," in *Proceedings of the 6th International Symposium on Smart Graphics*, 2006, pp. 70-81.
33. M. Weber, M. Alexa, and W. Muller, "Visualizing time-series on spirals," in *Proceedings of IEEE Symposium on Information Visualization*, 2001, pp. 7-13.
34. D. A. Gomez, R. Theron, and F. J. García-Peñalvo, "Semantic spiral timelines used as support for e-learning," *Journal of Universal Computer Science*, Vol. 15, 2009, pp. 1526-1545.
35. B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Proceedings of the 15th IEEE International Conference on Program Comprehension*, 2007, pp. 49-58.
36. J. García, A. González, D. A. Gómez, R. Therón, and F. J. García-Peñalvo, "A visual analytics tool for software project structure and relationships among classes," in *Proceedings of the 9th International Symposium on Smart Graphics*, 2009, pp. 203-212.
37. L. Voinea and A. Telea, "Multiscale and multivariate visualizations of software evolution," in *Proceedings of the 3rd ACM Symposium on Software Visualization*, 2006, pp. 115-124.
38. P. C. Wong and J. Thomas, "Visual analytics," *IEEE Computer Graphics and Applications*, Vol. 24, 2004, pp. 20-21.

Antonio González-Torres received his master degree in Computer Sciences from the University of Costa Rica in 2001 and currently he is pursuing the Ph.D. in Computer Sciences at the University of Salamanca, Spain. He has worked as network administrator for more than 12 years and holds industry certifications in the field. His current research interests include visual analytics, software evolution, network security and visual network analytics. Currently, he is member of the GRIAL (Research Group in InterAction and eLearning) and VisUsal research groups.

Francisco José García-Peñalvo received the Ph.D. degree in Informatics, Salamanca University in 2000. He is working in Informatics Department of the Salamanca University, Spain and leads the researching group GRIAL (research GRoup in InterAction and eLearning). He is member of the Science Education Research Institute. His current research interests include human-computer interaction, software engineering, semantic web, visual analytics, services oriented systems and technology management applied to educational contexts, systems and institutions.

Roberto Therón received his doctoral degree in Robotics at the University of Salamanca, where his research focused on combining fields such as computer science, artificial intelligence, statistics, and data visualization. Currently is the manager of the Visual Analytics and Information Visualization Research Group. His current research interests lie in the proposal of customized and advanced visual interfaces for specific analytical tasks in various domains (from bioinformatics to linguistics).