Computers in Human Behavior 29 (2013) 486–495

Contents lists available at SciVerse ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Human–computer interaction in evolutionary visual software analytics

Antonio González-Torres *, Francisco J. García-Peñalvo, Roberto Therón

Faculty of Sciences, Department of Computer Sciences, University of Salamanca, Spain

ARTICLE INFO

Article history:
Available online 26 February 2012

Keywords:
Evolutionary visual software analytics
Visual software analytics
Visual analytics
Human–computer interaction

ABSTRACT

Software evolution is made up of changes carried out during software maintenance. Such accumulation of changes produces substantial modifications in software projects and therefore vast amounts of relevant facts that are useful for the understanding and comprehension of the software project for making additional changes. In this scenario, evolutionary visual software analytics is aimed to support software maintenance, with the active participation of users, through the understanding and comprehension of software evolution by means of visual analytics and human computer interaction. It is a complex process that takes into account the mining of evolutionary data, the subsequent analysis of the mining process results for producing evolution facts, the use of visualizations supported by interaction techniques and the active participation of users. Hence, this paper explains the evolutionary visual software analytics process, describes a framework proposal and validates such proposal through the definition and implementation of an architecture.

1. Introduction

Visual analytics is a process whose goal is to provide insight into vast amounts of scientific, forensic, academic, or business data that are stored by heterogeneous data sources such as databases, html, and XML files. It iteratively collects information, preprocesses data, carries out statistical analysis, performs data mining, and uses machine learning, knowledge representation, user interaction, visual representations, human cognition (Drigas, Koukianakis, & Papagerasimou, 2011), perception, exploration and the human abilities for decision making (Keim, Mansmann, Schneidewind, & Ziegler, 2006; Keim, Mansmann, Schneidewind, Thomas, & Ziegler, 2008; Llorá, Sastry, Alías, Goldberg, & Welge, 2006).

In summary, visual analytics combines the advantages of machines with the strength of humans such as analysis, intuition, problem solving, and visual perception. Therefore, the human is at the heart of visual analytics (Dix, Pohl, & Ellis, 2010) and human computer interaction (HCI) is a key component for supporting knowledge discovery.

Accordingly, any visual analytics design should be centered on the user and intents to facilitate usability and reduce memory load on users (Hollender, Hofmann, Deneke, & Schmitz, 2010). Its ultimate goal should be to hide complexity details from users and provide an environment for knowledge discovery through an outstanding human experience (Takatalo, Nyman, & Laaksonen, 2008). Hence, regardless of the complexity of the problem at hand, the success of any visual analytics solution lies on the appropriate design of the visual representations and use of interaction techniques. This also applies to any visual analytics specialization such as visual e-learning analytics (Aguilar, Guerrero, Sánchez, & García-Penalvo, 2010), visual software analytics (Anslow, Noble, Marshall, & Tempero, 2009; Telea & Voinea, 2011; van den Brand, Roubtsov, & Serebreni, 2009), and evolutionary visual software analytics (González-Torres, Therón, Garcia-Penalvo, Wermelinger, & Yu, 2011).

Evolutionary visual software analytics is the process for supporting software maintenance, with the active participation of users, through the understanding and comprehension of software evolution by means of visual analytics and human computer interaction. It is a relatively recent application area of visual analytics to software evolution (González-Torres et al., 2011; Telea, Voinea, & Ersoy, 2010) and the explicit definition of such process has not been carried out yet. This makes it difficult to define roles and borders within the context of evolutionary visual software analytics using a common language. Overall, it affects the communication and understanding of researches by other scholars, as well as conducting new research that could easily be coupled to previous research. As a result, the lack of such definition could limit the development and planning of modular evolutionary visual software analytics tools, producing an impact in integration and reusability of components.

The explicit definition of the process could support the understanding of evolutionary visual software analytics through the use of a common language. In addition, the definition of roles, borders and relationships between research areas, methods and techniques can be planned and carried out taking into consideration

* Corresponding author.
E-mail addresses: agtorres@usal.es (A. González-Torres), theron@usal.es (F.J. García-Peñalvo), fgarcia@usal.es (R. Therón).

0747-5632/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.chb.2012.01.013

the overall process. Furthermore, the elements of the process can be considered as components, where each component has to support the function of other components and produce an output that can be used as the input of other components. Therefore, the use of the process definition could make it easier to identify the interactions of components and their localization within the process, when used as a location map.

The benefits of defining the evolutionary visual software analytics process also apply to visual software analytics, visual e-learning analytics and visual analytics due to the absence of specific process definitions for those research fields. Hence, this paper concerns with the explicit definition and validation of a framework to describe the evolutionary visual software analytics process and identify the involved research areas, methods, techniques, and relationships among them. Afterwards, it takes into consideration the central role played by users in knowledge discovery tasks. Furthermore, it discusses the validation of the framework by means of a tool that has been developed upon an architecture based on such framework.

Consequently, the remainder of this paper is organized as follows: Section 2 describes the visual analytics process for providing the basics of the following sections, Section 3 makes an introduction to Software maintenance and evolution, evolutionary visual software analytics and the role played by users, and Section 4 discusses the visual software analytics process. Furthermore, Section 5 discusses the evolutionary visual software analytics framework, Section 6 explains an evolutionary visual software analytics architecture based on the framework proposal, Section 7 addresses the visualizations and interaction design characteristics, Section 8 explains two use case scenarios of the tool and finally Section 9 discusses the main conclusions.

2. The visual analytics process

Research papers in visual analytics frequently focus on the discussion of the analytical process carried out with the use of linked interactive visual representations as well as the design and features of such visual representations. Therefore, usually it is not taken into consideration that visual analytics is a broader process which also encompasses the extraction of information from heterogeneous data sources, the analysis of such information and the representation of the outcome using abstract data structures.

The visual analytics process described in Fig. 1 has been carried out after a careful review and analysis of some of the most influential application frameworks, tool architectures and definitions that have been published upon the visual analytic process (Keim et al., 2006; Keim, Kohlhammer, Ellis, & Mansmann, 2010; Shneiderman, 1996; Thomas & Cook, 2005). The main tasks of this process 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.

The visual analytics process begins with the reading of data from heterogeneous data sources. Then, the knowledge extraction engine uses one or more knowledge extraction techniques, including but not limited to the ones shown in Fig. 1, for discovering, representing and managing knowledge. Next, the data transformation engine for visualization models takes the output of the knowledge extraction engine (first analysis results, arrow 3) and creates the appropriate data structures required by the visual knowledge explorer to display the most important results (show the important, arrow 4). The visual knowledge explorer uses a combination of techniques from information visualization, usability, information

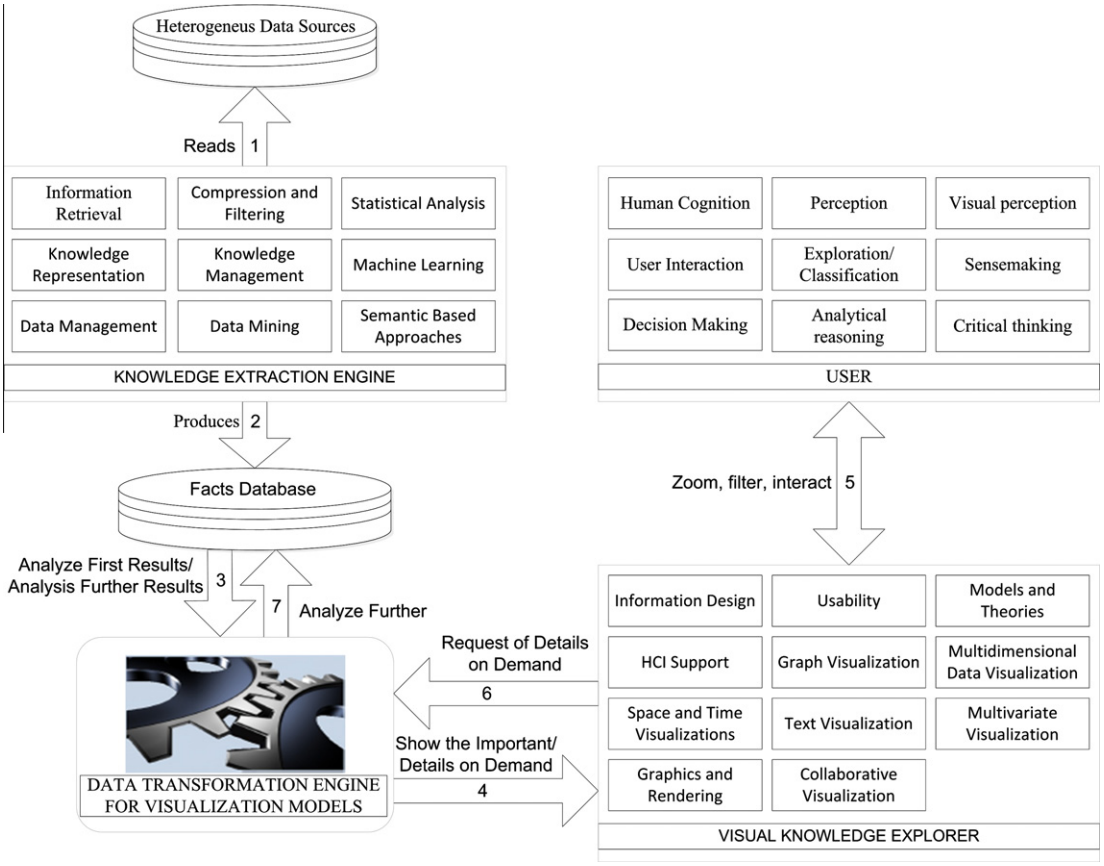

Fig. 1. The visual analytics process.

design, and human computer interaction to accomplish its main task; provide a means to users for knowledge exploration and discovery through the use of interaction, human perception and cognition for decision making (zoom, filter, interact, see arrow 5).

At this point the visual analytics process becomes an iterative process that could require 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 6). Hence, if the visual knowledge explorer requests additional information for providing more details to users, and the requested data are unavailable in the persistent data structures created by the data transformation engine, the knowledge extraction engine performs further analysis tasks (further analysis, arrow 7). Then, the results are passed to the data transformation engine and added to the persistent data structures (further analysis results, arrow 3). Next, the corresponding details are visually represented in the visual knowledge explorer (details on demand, arrow 4) and the user continues working on the knowledge discovery process. This iterative process stops when the user has:

1. arrived to useful conclusions or
2. decided that further manual analysis is needed for uncovering additional details and being able to arrive into conclusions or
3. considered that drawing conclusions from the available knowledge, visual representations or data is unfeasible.

3. Background

Software developers and managers often faced the maintenance of large legacy applications and software projects to which they usually do not provide support, either within their company or a client company. Understanding the evolution of such legacy applications or software projects is a crucial task. Particularly because software maintenance is a usually compromised process due to the lack of proper system documentation, which frequently is incomplete, outdated or not present (Murphy & Notkin, 1997). The challenge of understanding a software project is to gain insight on the effects produced by changes made to the project during its maintenance. Subsequently, comprehension of the effects caused by changes allows the implementation of the appropriate actions to make additional changes if the quality or functionality of the software project is compromised in the short or long term.

Accordingly the next sections deal with the discussion of software maintenance and evolution, the evolutionary visual software analytics process and user's analytical tasks by means of human–computer interaction.

3.1. Software maintenance and evolution

Software development and maintenance are dynamic tasks that conform to the basis of software evolution. Software maintenance starts from the moment in which a software application has been conceived and its design has started. During software maintenance, project managers and developers necessitate the understanding of the software project structure and its source code, the relationships among software items and software quality metrics for making preventive (for improving source code quality), adaptive (due to software functionality improvements or additional requirements) and corrective changes. In this context, a software item is a source code piece such as a class or interface and the relationships among software items are those defined in the object oriented programming theory as well as other relationships, including the direct and indirect coupling.

Software maintenance is a cyclic process; changes are based on the understanding of the current state of the software project, which is the accumulation of previous changes made by the software maintenance activity. The change process and the tracking of changes are usually managed with the assistance of a software configuration management tool (SCM).

A SCM tool uses revisions for storing details about changes, such as the author who made the change, the date and time of the change, the project structure before and after the change, and the source code and the changes that were carried out on it (Estublier, 2000).

A revision identifies the current state of the project at the moment that the change has been committed. Revisions are stored by a software repository under the control of a SCM tool and are associated to a revision number. Consequently, software evolution is an iterative process conformed by the accumulation of changes and revisions during software maintenance and development (Fernandez-Ramil, Lozano, Wermelinger, & Capiluppi, 2008). Understanding a revision includes the comprehension of the structural characteristics of the project, the relationships among software items, the software quality metrics, source code facts, and the comprehension of the socio-technical relationships derived from the development process. The challenge of understanding the evolution of a software project is to gain insight on the effects produced by changes made to the project. Therefore, comprehension of the effects caused by changes allows the implementation of the appropriate actions to make additional changes if the quality or functionality of the software project is compromised in the short or long term.

3.2. Evolutionary visual software analytics

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 comprehension process by interacting with the visual representation.

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 discovery process and comprehension of relevant facts regarding the evolution of the software project.

In addition, it is important to consider that the analysis process is very complex because the life cycle of large software projects usually expands through several years, generates thousands and even millions of lines of source code (LOC) (Kagdi, Collard, & Maletic, 2007), hundreds of software components and thousands of revisions (DAmbros, Gall, Lanza, & Pinzger, 2008). Furthermore, within software projects exist relationships among software items in the form of inheritance, interface implementation, coupling and cohesion. In addition, source code is made up by variables, constants, programming structures, methods and relationships among those elements. Besides logs, communication systems, defect-tracking systems and SCM tools keep records with dates, comments, changes made to software projects and associated users and programmers (Hassan, 2005).

Accordingly, the analysis of individual revisions and the comparison of the analysis performed on them are carried out by the mining of software repositories on an evolutionary basis. The data used by software repository mining tools are collected from the source code, the software project structure, communication systems, logs and the metadata records from bug tracking and SCM

tools. Thus, an important consideration is that the proposal makes reference to the mining of software repositories on an evolutionary basis as the use of any set of extraction and analysis techniques that have the capability of extracting and analyzing software projects looking to discover patterns and relationships and calculating software quality metrics and fact extraction from the results of comparing the analysis performed on revisions (DAmbros et al., 2008).

Moreover, the datasets produced by mining software repositories on an evolutionary basis are usually overwhelming, due to their large size, and are hard to understand by humans. This makes it necessary to provide a means for allowing software project managers and programmers to get insight and grasp the details of the project evolution. Therefore, information visualization techniques provide such a means with the support of interaction techniques and linked views. However, before visually representing the output generated by the mining of software repositories, such output needs to be transformed into the appropriate data structures that will be used as the input of the visual representations.

A summary of this process is that the evolution of software projects produces a wealth of evolutionary data that is stored in software repositories (Hassan, 2005), communication systems, logs and bug tracking databases, which are then mined to discover patterns, relationships and trends (DAmbros et al., 2008). Finally, data are transformed into the appropriate data structures to be turned into an opportunity (Keim et al., 2006) by means of visual representations.

Hence the application of visual analytics to software systems and related processes, with the aim of supporting the understanding of a software project or an individual revision, is known as visual software analytics (Anslow et al., 2009; Telea & Voinea, 2011; van den Brand et al., 2009). Accordingly, this research defines the process described above as evolutionary visual software analytics.

3.3. Human–computer interaction

The visual representation of huge datasets supported by browsing and navigation capabilities is challenging due to the limited size of screens (Leung & Apperley, 1994). From there, the design of visual representations could be carried out taking into consideration a spatial or temporal design strategy or even better by combining both. On one hand, the spatial design strategy layouts graphical elements for representing the information in one view. While on the other hand, the temporal design strategy makes use of transitions among views for distributing the information in multiple views (Mackinlay, Robertson, & Card, 1991). It uses linked views and interaction mechanisms to allow the exploration of details while keeping the context (Hornbaek & Hertzum, 2011; Rao & Card, 1994) and allows the tracking of relationships to facilitate the interpretation of specific elements.

In this context, one should to take into consideration that analysts are strongly influenced by their experience, education, cultural values (Richards & Heuer, 1999) and cognition (Drigas et al., 2011) that allows them to gradually acquire strategies for remembering, understanding, and solving problems (Academies, 2000). An interesting reference on this is the framework on information behavior proposed by Spink using an evolutionary perspective (Spink, 2010). In addition to some relevant researches such as the one carried by Pirolli on visual information foraging (Pirolli, Card, & Wege, 2001), which deals with visual attention and information foraging theory. Where the latter is concerned with search, exploration, location and evaluation of information (Chen, Cribbin, Kuljis, & Macredie, 2002).

Therefore, one can think that users form a hypothesis to solve a problem, then collect data, analyze such data and then accept or reject the initial hypothesis. From there, several researches have been conducted regarding how users browse and navigate through visual representations looking to solve complex problems by means of visual analysis. One of the most notable researches in this field is summarized by the Shneiderman's visual information seeking mantra (overview first, zoom and filter, then details-on-demand) that outlines the tasks usually performed by users when navigating information visualization designs (Shneiderman, 1996). The results of such research has been reinforced by several authors, such as the visual information seeking mantra's adaptation made by Keim for reflecting the tasks carried by visual analytics (analyze first, show the important, zoom, filter, and analyze further, details on demand) (Keim et al., 2006). Additionally, Wehrend et al. have defined a taxonomy of user tasks in visual environments that recognizes the following eleven actions: identify, locate, distinguish, categorize, cluster, distribution, rank, compare within relations, compare between relations, associate and correlate (Wehrend & Lewis, 1990).

Accordingly, interaction design patterns has been designed for general repeatable solutions in interface and interaction design (Pauwels, Hbscher, Bargas-Avila, & Opwis, 2010; Tidwell, 2011) and several researches has been conducted on visualization design and human computer interaction.

The next sections focus on discussing the visual software analytics and evolutionary visual software analytics processes and make comparisons between them and visual analytics. The intent of having discussed visual analytics first and then concentrating in visual software analytics and evolutionary visual software analytics, in that order, is to show an evolution in the process and provide a better explanation for the apprehension of evolutionary visual software analytics.

4. The visual software analytics process

Visual software analytics is based upon the visual analytics process and it is aimed to the study and analysis of the dynamics of software systems (see Fig. 2). Its main tasks have similar analytic goals to their analogs in visual analytics. Although each task is made of techniques and methods that focus on knowledge discovery from software facts.

On the other hand, a practical analogy for explaining the difference between visual software analytics and evolutionary visual software analytics is that the first is a movie frame while the latter is a movie. Thus, visual software analytics takes into account only a revision of a software project whereas evolutionary visual software analytics takes into account all the revisions of the software project. Consequently, both processes share several techniques although the evolutionary approach also requires an extra analysis for comparing the results carried out on individual revisions.

Accordingly, understanding visual software analytics helps to comprehend evolutionary visual analytics as both processes share the same sequence of tasks and have similar aims, even though they differ with regards to some specific methods and techniques used. Hence, some data sources in common are SCM[1] repositories metadata, defect-tracking logs, email communications, and source code.

The main components of visual software analytics are the knowledge extraction engine, the software facts database, the data transformation engine for visualization models, the visual knowledge explorer for software visualization and the user.

The knowledge extraction engine reads software project associated data from the aforementioned data sources for extracting software facts. When the knowledge extraction engine has carried out the analysis, it stores the software facts in its corresponding data-

[1] Software configuration management.

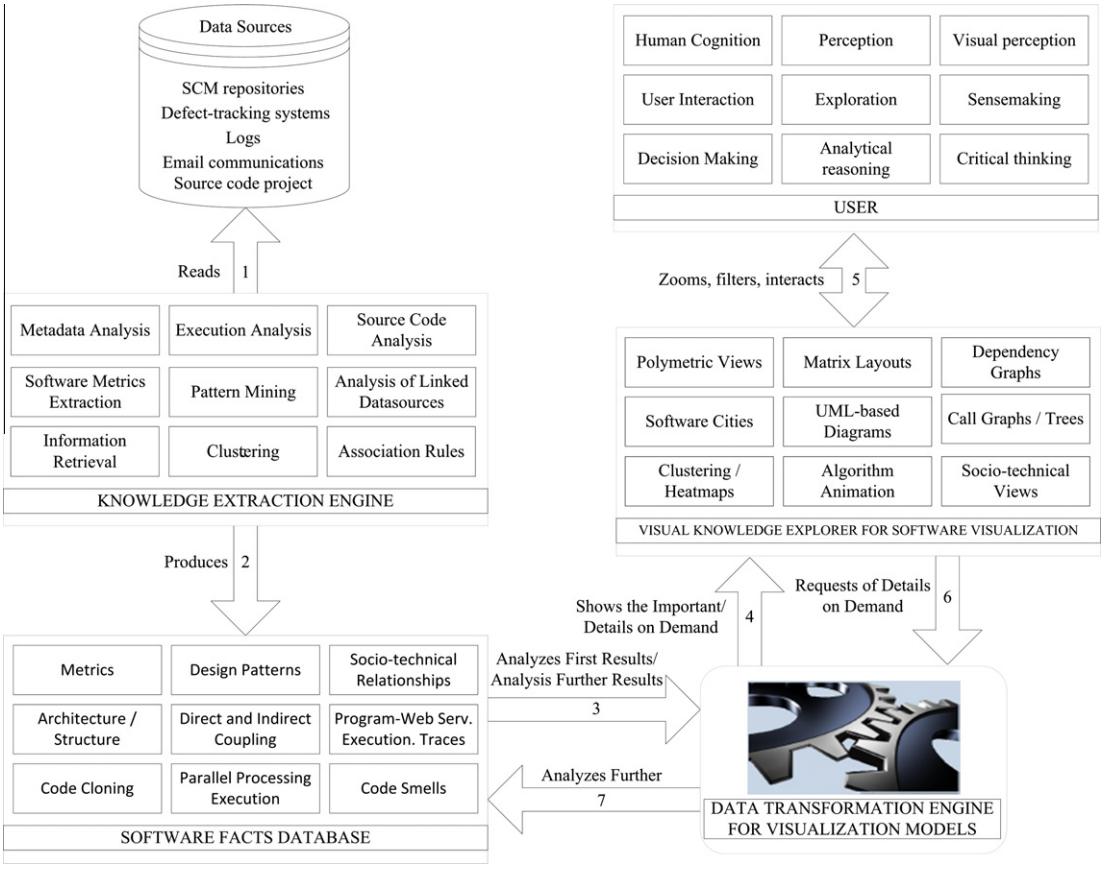

Fig. 2. The visual software analytics process.

base. Therefore, the data transformation engine for visualization models transforms the software facts data structures into the appropriate data structures required by the visual knowledge explorer for software visualization.

Finally, the software facts are represented by means of software visualization techniques and the user starts the interaction process looking to understand the software project and get insight of the software project.

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 play a relevant role in the knowledge discovery process. Programmers and project managers will 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 process stops upon the user's decision.

5. A framework for the evolutionary visual software analytics process

Unlike previous sections, this section is not going to deal with the description of the evolutionary visual software analytics process (see Fig. 3) because it is reiterative with relation to visual analytics and visual software analytics, as all of them use the same sequence of tasks. Hence, this section will encompasses a discussion of the methods and techniques employed by evolutionary visual software analytics as well as the understanding and comprehension goals it looks to achieve.

Evolutionary visual software analytics supports the understanding of how software projects evolve and provides details on how software item relationships have been established. Its goals usually include software quality assurance, the improvement of the software maintenance process, the forecasting of project related events and assisting software developers in the understanding of the software project, aiding software project managers in resource management, including programmer related tasks.

The complexity of evolutionary visual software analytics is higher with respect to 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. As a consequence, the design of the visualizations is confronted with the layout of time and space and the user deals with a more complex scenario. Therefore, users become the central player of the understanding process, in which the use of its capabilities and human interaction are the key to disclosing the intricacies of software projects complexity.

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, classification change, defect tracking and communications, and architectural relationships. For this purpose it makes use of several analysis techniques, among which are origin analysis, contribution analysis, software prediction models, frequent coupling mining, evolutionary metadata analysis, frequent patterns mining, refactoring analysis, source

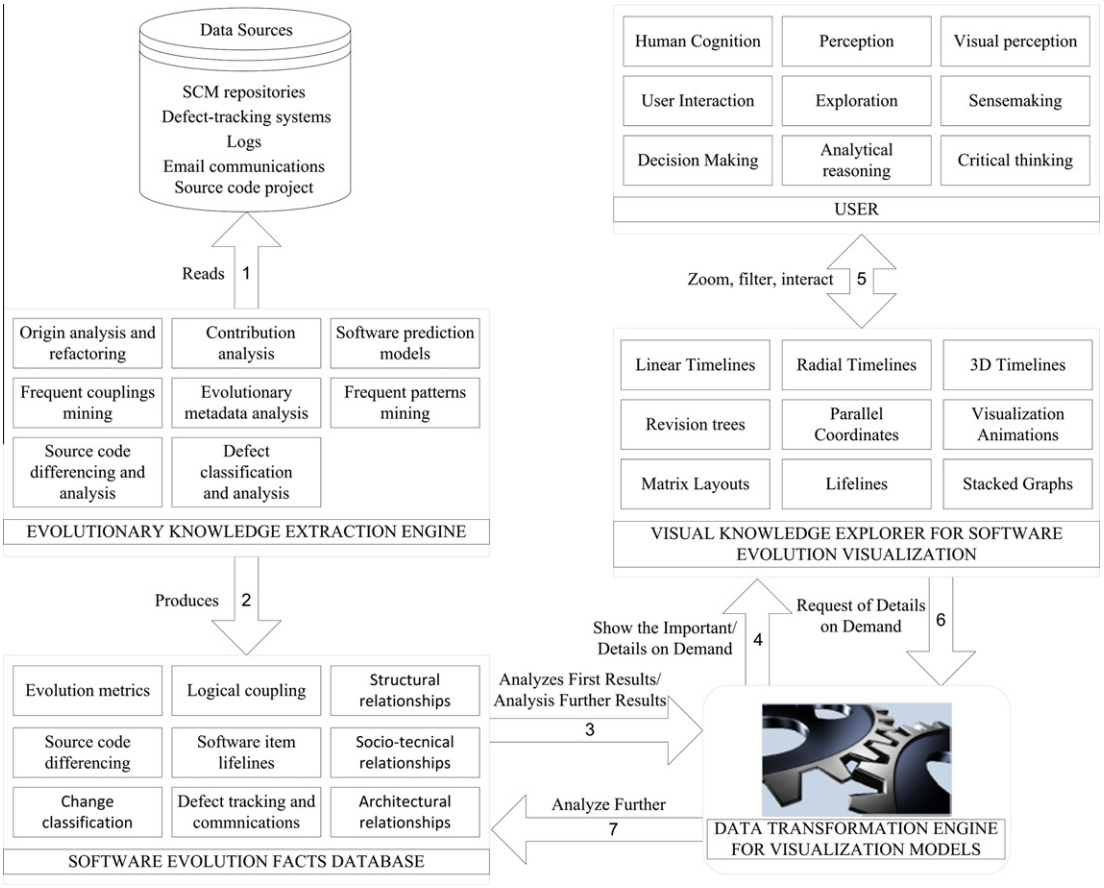

Fig. 3. The evolutionary visual software analytics process.

code differencing and analysis and defect classification and analysis.

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

6. An evolutionary visual software analytics architecture

Defining the architecture of software tools is a rather complex task that requires careful analysis. It is a challenge to determine what techniques to use, how these will relate to each other and contribute to the research or the tool design goals. This paper is aimed to provide a framework to novel researchers and those designing tool architectures in scenarios where visual analytics is applied to software evolution. Previous sections have so far discussed how evolutionary visual software analytics contributes to software maintenance and the definition of a framework for the evolutionary visual software analytics process. This section takes as reference the framework presented in Section 5 for defining the architecture of a tool that supports the understanding and comprehension of software projects during software maintenance.

The tool architecture is shown in Fig. 4 and its implementation has been carried out in Java. Such architecture has been tested on 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 supporting 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 monitor new revisions, the source code extractor, the source code parser and the metadata and software evolution analysis engine. 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 query 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 software evolution facts that have been taken into account for the visualizations presented later 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 data structures into the appropriate data structures that are used by software evolution visualizations. Therefore, the visualizations are embedded into an Eclipse plugin and make use of a matrix-like representation, a timeline and socio-technical views that are supported by a social-network graph and the use of colors for representing programmer's contributions.

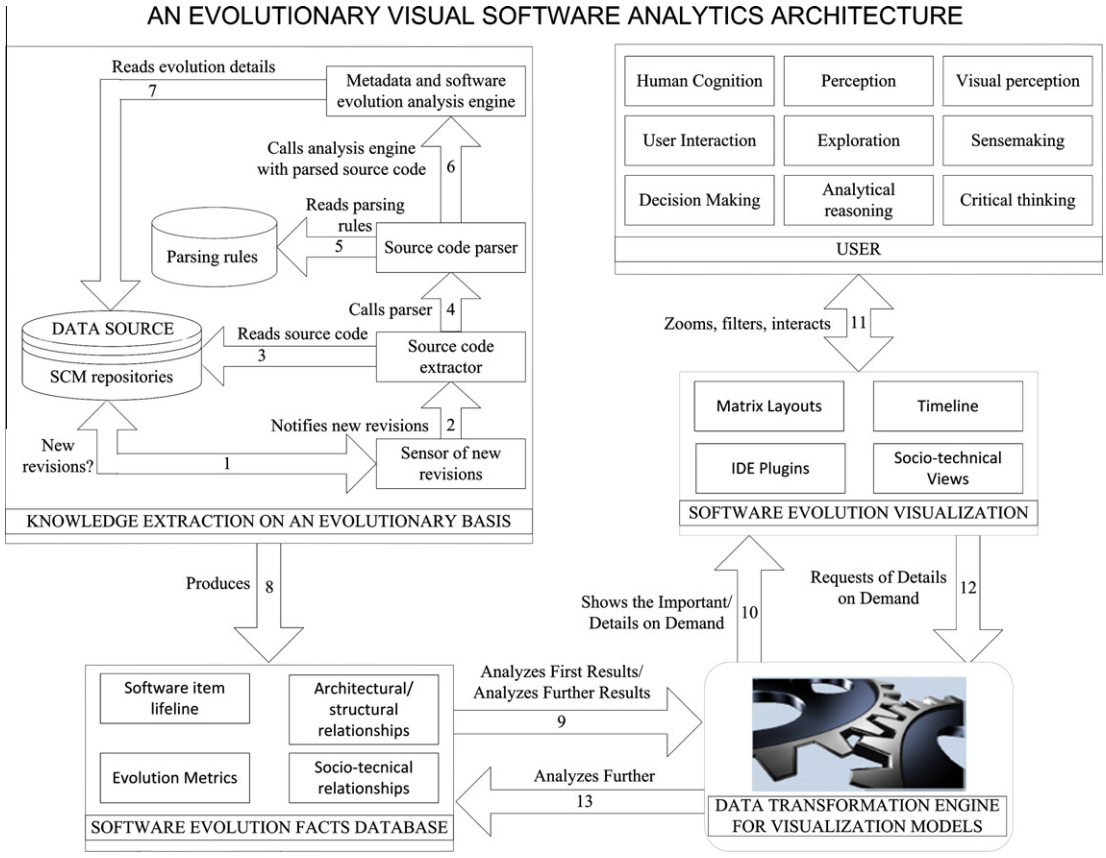

Fig. 4. The evolutionary visual software analytics process.

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

7. Visualizations and interaction design

A matrix-like representation has been chosen as the main view for being a simple structure widely known by programmers and because it allows establishing relationships among multivariate elements easily (see Fig. 5). 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.

The interactions supported by the matrix view include 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 and in the socio-technical view. Moreover, the user has the possibility to choose how metrics and programmers are represented by selecting between relative or absolute value representations.

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 to correlate all software items involved in the evolution process with programmers, metrics and architectural relationships such as inheritance and interface implementation relationships. Fig. 6a shows the current project structure of jEdit on the left in the Eclipse workbench, while on the right the visual representation is displayed depicting the project and its corresponding structure, including packages and software items that are no longer part of the current project version. Additionally, Fig. 6b illustrates that source code files could be expanded for showing the software items they contain. Such design feature allows to depict the lifeline of software items and packages using an intuitive approach, as shown in Fig. 5. The details view on the right shows that the performed activities on the file macros. Delegate.java were carried out between 2003 and 2008. Moreover, those packages are not currently part of the latest version of the project, which is corroborated when reviewing the project structure on the Eclipse workbench, as shown in Fig. 6a. So, the lifeline of a package or software item is determined by the representation of programmer activity in the matrix view.

Another visualization key feature is the representation of inheritance and interface implementation, which is depicted in Fig. 6c. Foldable tree nodes are used for representing software item inheritance and implementation relationships. Fig. 7 shows that the establishment of inheritance and interface implementation relationships are depicted 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 (see Fig. 8). Similar to the representation of programmer contributions, metrics are represented using relative and absolute areas. Relative representation takes under consideration the software item with the highest metric value for calculating the chart height, while the absolute representation only takes under consideration the highest metric value associated to the software item.

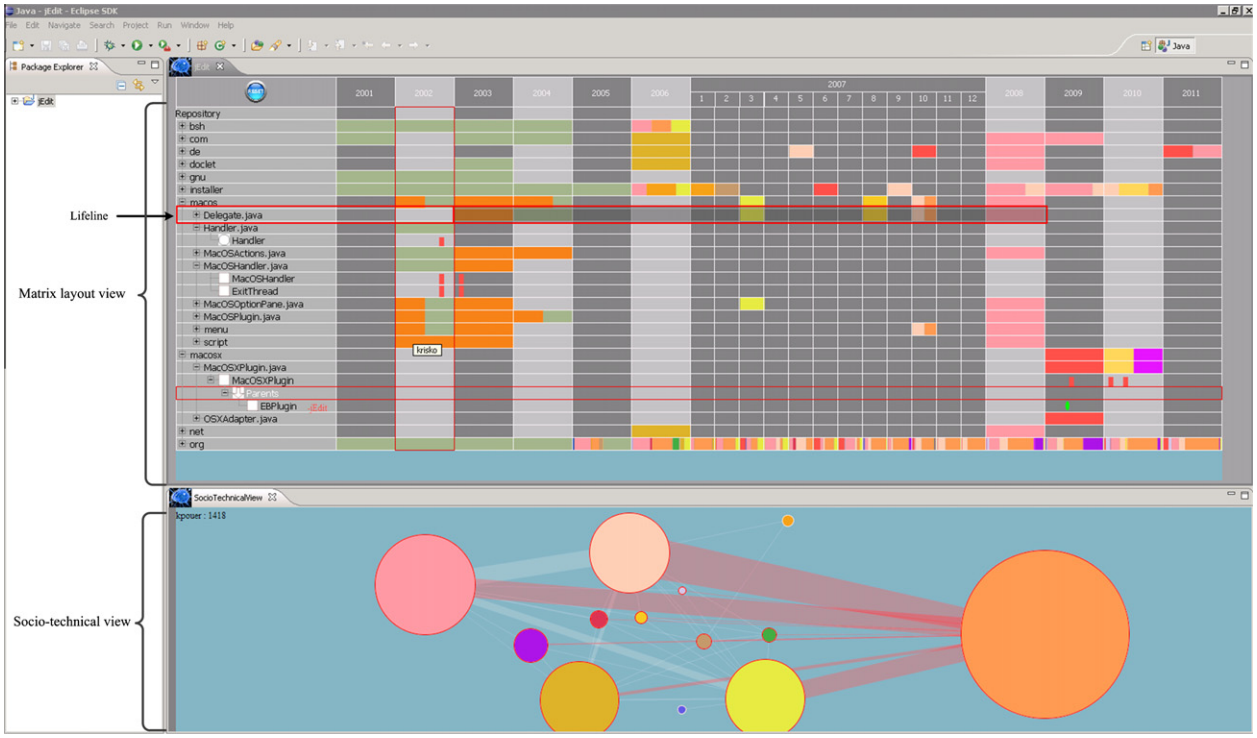

Fig. 5. Main view of the visualizations.

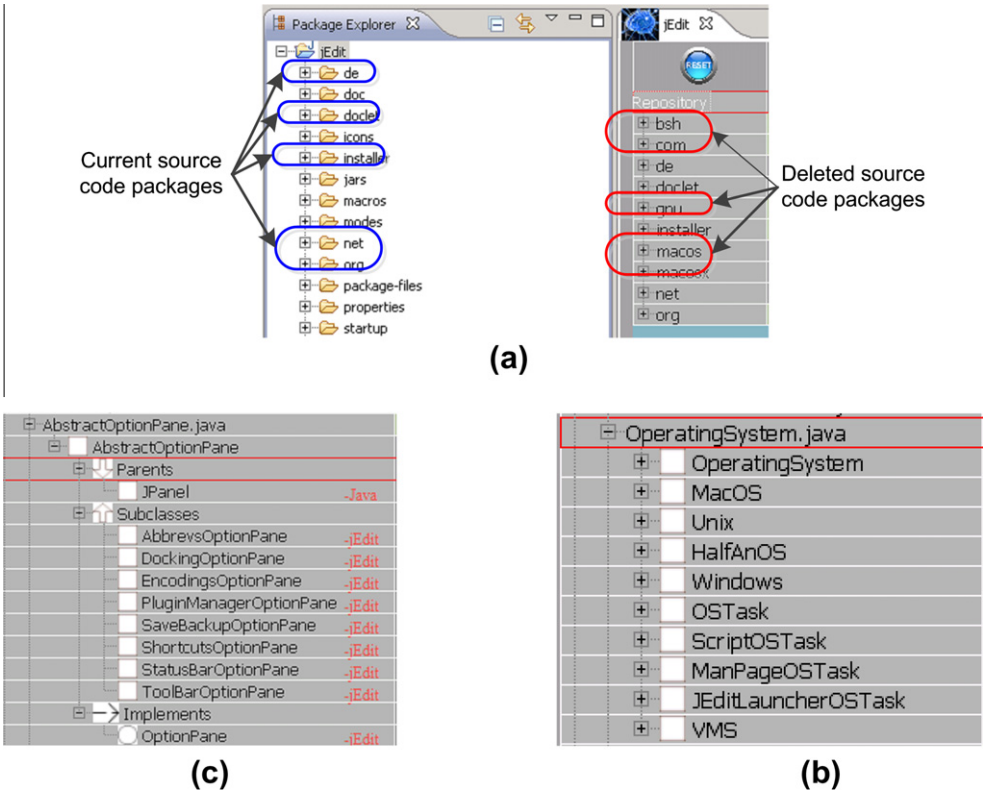

Fig. 6. Project structure and file contents (a) project structure representation, (b) software items contained by a file, and (c) inheritance and interface implementation relationships.

The other included view in the visual representation design is the socio-technical view (see Fig. 5), which is subordinated to the matrix view. When users select a time unit in the matrix view the associated data are loaded into the socio-technical view. Loaded data represent the socio-technical relationships that derive from the modification of software items. Such relationships are

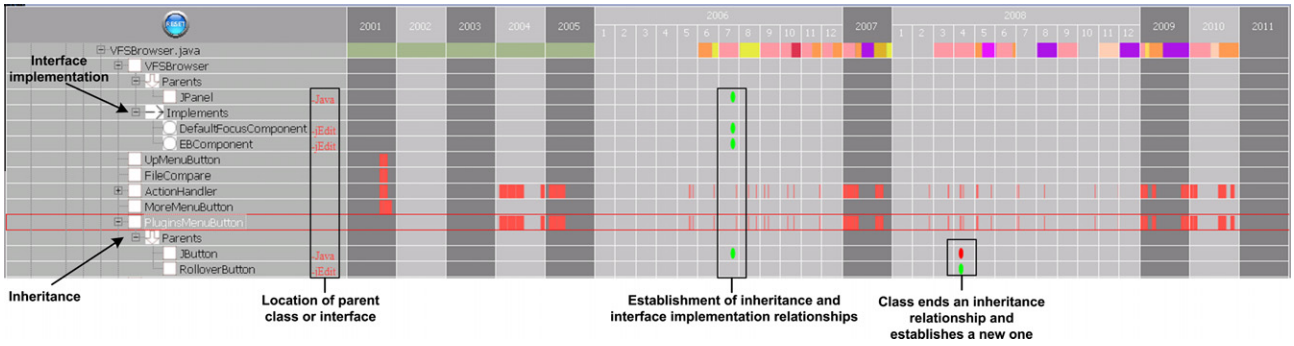

Fig. 7. Inheritance and interface implementation relationships, including expanded years, and metric values.

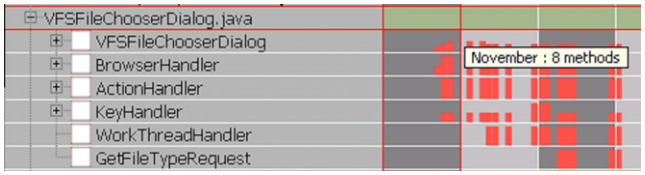

Fig. 8. Metrics representation.

established from the software items that programmers have modified in common. The nodes of the graph represent to programmers and their size represent the number of contributions they have been made. In addition, edges represent the relationships among programmers and their thickness represent the number of software items they have modified in common. Node colors represent programmers and are associated to the colors used in the matrix view.

8. Use case scenarios

The visual representations have been integrated into Eclipse as a plugin, shown in Fig. 5.

The interaction path, followed by users, starts with the selection, from a popup menu, of the view that they want to display in the Eclipse workbench. From this point users interact with the visualization and the interaction path branches. So, several scenarios are possible according to maintenance needs. The following scenarios are some practical examples in which the visualizations could be applied.

Use case 1: A bug has been reported and the project manager has to assign a programmer to solve it. Once the project manager has analyzed who is the most suitable programmer for solving the reported issue, the project manager realizes that the developer in charge of the components in which the problem is localized is on vacation. Then, the project manager follows the steps below:

1. Open the matrix view for the complete project or a given package and then the project manager opens the socio-technical view.
2. Select a recent time unit, with a considerable number of activities, from the timeline in the matrix view.
3. The socio-technical view is updated with the data associated to the selected time unit.
4. The project manager selects the programmer that is on a sick leave from the socio-technical view and the visualization highlights the relationships of such programmer with the programmers that have changed the same software items.
5. Then, the project manager determines which programmer is the most suitable for solving the opened issue.

Use case 2: Programmers are in the middle of a software project refactoring and they have to review how changes made by other programmers affect inheritance and interface implementation relationships. Programmers follow the next steps after opening the matrix view for the complete project or a given interesting package:

1. Choose the source code file, from the matrix view project navigator, that contains the software item in which the programmer is interested and expand it to review the relationships of such software item.
2. Expand the software item and examine its inheritance and interface implementation relationships to determine if changes have occurred: which relationships have been established and which have been terminated.

A video demo of the tool prototype is available from http://www.analiticavisual.com/maleku.

9. Conclusions

Visual analytics is an emerging discipline that it is applied to many human knowledge fields and therefore it has been applied to software evolution until recently. Consequently, the main contributions of this paper are the definition of the evolutionary visual software analytics process, an evolutionary visual software analytics framework and its validation through the implementation of a tool using it as reference. Such validation has proven the feasibility of the application of the framework to research and tools design.

The definition of the evolutionary visual software analytics process and its framework support the understanding of evolutionary visual software analytics through the use of a common language. In addition, the definition of roles, borders and relationships among research areas, methods and techniques can be planned and carried out taking into account the overall process. Furthermore, the elements of the process can be considered as components; where each component has to support the function of other components and produce an output that can be used as the input of other components. Therefore, the benefits of defining the aforementioned process also apply to visual analytics and visual software analytics, due to the detailed explanation that has been provided regarding those processes.

In addition, this paper has made emphasis in the role played by users and have discussed a visualization design that has taken into account several needs to be informed about the evolution of software projects. Some concern which programmers are participating in the development of specific part of a project, others concern when the project has taken place and when the solution becomes stable. It also can inform the software maintainers about which programmer has created more revisions and new files, and how

developers are collaborating or working separated from one another

References

Academies, T. N. (2000). *How people learn: brain, mind, experience, and school*. National Academy Press.

Aguilar, D. A. G., Guerrero, C. S., Sánchez, R. T., García-Penalvo, F. (2010). Advances in learning processes. *InTech. chapter visual analytics to support e-learning*, pp. 207–228.

Anslow, C., Noble, J., Marshall, S., Tempero, E. (2009). Towards visual software analytics. In *Proceedings of the Australasian computing doctoral consortium (ACDC)*. Wellington, New Zealand.

Chen, C., Cribbin, T., Kuljis, J., & Macredie, R. (2002). Footprints of information foragers: Behaviour semantics of visual exploration. *International Journal of Human–Computer Studies, 57*, 139–163.

DAmbros, M., Gall, H. C., Lanza, M., Pinzger, M. (2008). Analyzing software repositories to understand software evolution. In *Software evolution*.

Dix, A., Pohl, M., & Ellis, G. (2010). Mastering the information age solving problems with visual analytics. *Eurographics Association*, 109–130. chapter Perception and Cognitive Aspects.

Drigas, A., Koukianakis, L., & Papagerasimou, Y. (2011). Towards an ict-based psychology: E-psychology. *Computers in Human Behavior, 27*, 1416–1423.

Estublier, J. (2000). Software configuration management: A roadmap. In *Proceedings of the conference on the future of software engineering* (pp. 279–289). New York, NY, USA: ACM.

Fernandez-Ramil, J., Lozano, A., Wermelinger, M., Capiluppi, A. (2008). Software evolution. *Empirical studies of open source evolution* pp. 263–288.

González-Torres, A., Therón, R., Garcia-Penalvo, F. J., Wermelinger, M., Yu, Y. (2011). Maleku: an evolutionary visual software analytics tool for providing insights into software evolution. In I.C. Society (Ed.), *IEEE international conference on software maintenance (ICSM)*.

Hassan, A. E. (2005). Mining software repositories to assist developers and support managers. Ph.D. thesis. Waterloo, Ont., Canada, Canada.

Hollender, N., Hofmann, C., Deneke, M., & Schmitz, B. (2010). Integrating cognitive load theory and concepts of human computer interaction. *Computers in Human Behavior, 26*, 1278–1288.

Hornbaek, K., & Hertzum, M. (2011). The notion of overview in information visualization. *International Journal of Human–Computer Studies, 69*, 509–525.

Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). 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, 19*, 77–131.

Keim, D., Kohlhammer, J., Ellis, G., Mansmann, F. (2010). Solving problems with visual analytics. In *Eurographics Association*.

Keim, D. A., Mansmann, F., Schneidewind, J., Thomas, J., & Ziegler, H. (2008). *Visual data mining*. Berlin, Heidelberg: Springer-Verlag. chapter Visual Analytics: Scope and Challenges.

Keim, D. A., Mansmann, F., Schneidewind, J., & Ziegler, H. (2006). Challenges in visual data analysis. *IV '06, Proceedings of the conference on information visualization* (pp. 9–16). Washington, DC, USA: IEEE Computer Society.

Leung, Y. K., & Apperley, M. D. (1994). A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions in Computer Human Interaction, 1*, 126–160.

Llorá, X., Sastry, K., Alías, F., Goldberg, D. E., & Welge, M. l. (2006). Analyzing active interactive genetic algorithms using visual analytics. *GECCO '06, proceedings of the 8th annual conference on genetic and evolutionary computation* (pp. 1417–1418). New York, NY, USA: ACM.

Mackinlay, J. D., Robertson, G. G., & Card, S. K. (1991). The perspective wall: detail and context smoothly integrated. *CHI '91: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 173–176). New York, NY, USA: ACM Press.

Murphy, G. C., & Notkin, D. (1997). Reengineering with reflection models: A case study. *IEEE Computer, 30*, 29–36.

Pauwels, S. L., Hbscher, C., Bargas-Avila, J. A., & Opwis, K. (2010). Building an interaction design pattern language: A case study. *Computer in Human Behaviour, 26*, 452–463.

Pirolli, P., Card, S. K., & Wege, M. M. V. D. (2001). Visual information foraging in a focus + context visualization. *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 506–513). New York, NY, USA: ACM.

Rao, R., Card, S. K. (1994). The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the SIGCHI conference on human factors in computing systems: Celebrating interdependence*, pp. 318–322.

Richards, J., Heuer, J. (1999). Psychology of intelligence analysis. United States Government Printing.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE symposium on proceedings of visual languages*. pp. 336–343.

Spink, A. (2010). *Information behaviour: An evolutionary instinct*. Heidelberg: Springer.

Takatalo, J., Nyman, G., & Laaksonen, L. (2008). Components of human experience in virtual environments. *Computers*, 1–15.

Telea, A., Voinea, L., Ersoy, O. (2010). Visual analytics in software maintenance: Challenges and opportunities. In D. Keim, J. Kohlhammer, (Eds.), *Proceedings of the 1st European Symposium on Visual Analytics (EuroVAST), Eurographics*.

Telea, A., & Voinea, L. (2011). Visual software analytics for the build optimization of large-scale software systems. *Computational Statistics, 26*, 635–654.

Thomas, J. J., & Cook, K. A. (2005). *Illuminating the path: Research and development agenda for visual analytics*. IEEE-Press.

Tidwell, J. (2011). Designing interfaces. *2nd ed.*. O'Reilly Media.

van den Brand, M., Roubtsov, S., Serebreni, A. (2009). Squavisit: A flexible tool for visual software analytics. In *CSMR 13th European conference on software maintenance and reengineering*. pp. 331–332.

Wehrend, S., Lewis, C. (1990). A problem-oriented classification of visualization techniques. In *Visualization '90. Proceedings of the first IEEE conference on visualization*, pp. 139–143, 469.