Knowledge discovery in software teams by means of evolutionary visual software analytics†

Antonio González-Torres[a], Francisco J. García-Peñalvo[b], Roberto Therón-Sánchez[b] and Ricardo Colomo-Palacios[c]

[a]School of Engineering, ULACIT, Costa Rica
[b]Department of Computer Sciences, University of Salamanca, Spain
[c]Faculty of Computer Sciences, Østfold University College, Norway

Abstract

The day-to-day management of human resources that occurs during the development and maintenance process of software systems is a responsibility of project leads and managers, who usually perform such a task empirically. Moreover, rotation and distributed software development affect the establishment of long-term relationships between project managers and software projects, as well as between project managers and companies. It is also common for project leads and managers to face decision-making on human resources without the necessary prior knowledge. In this context, the application of visual analytics to software evolution supports software project leads and managers using analysis methods and a shared knowledge space for decision-making by means of visualization and interaction techniques. This approach offers the possibility of determining which programmer has led a project or contributed more to the development and maintenance of a software system in terms of revisions. Moreover, this approach helps to elucidate both the software items[1] that have been changed in common by a group of programmers and who has changed what software items. With this information, software project leads and managers can make decisions regarding task assignment to developers and staff substitutions due to unexpected situations or staff turnover. Consequently, this research is aimed at supporting software practitioners in tasks related to human resources management through the application of Visual Analytics to Software Evolution.

Keywords: Visual Analytics, Software Evolution, Software Evolution Visualization, Visual Software Analytics, Evolutionary Visual Software Analytics

1. Introduction

The current tendency to perform the development of software systems in a distributed form by means of the so called distributed software development (DSD) must be considered when evaluating software processes[1–3]. DSD scenarios normally impede fluid communication and the construction of a shared knowledge space, which facilitates the understanding of the state of the project[4,5].It must be said that, without the construction of the aforementioned shared knowledge space, it is not possible to efficiently manage resources nor assign tasks appropriately, and neither is it possible to adequately manage collaboration between team members, manage the evolution of quality control and remain informed of all the activities carried out by members of the group [6].

The construction of a shared knowledge space and the fluidity of communication may be affected not only by geographical distance but also by cultural factors, which can result in less efficient communication between DSD work teams[7,8]. Additionally, a high level of activity and the technical complexity of much of these activities further limit communications, as it is not feasible, usually for reasons of time, to communicate either orally or in writing all the details of the work that is carried out. Thus, to construct a shared knowledge space, communications between work teams are not sufficient; neither is the use of a computer system in which details of the activities carried out could be reported by means of completing forms [9]. The result is that automated mechanisms that report the changes made and the tasks carried out are frequently put in place by means of Integrated Development Environments (*IDEs*) or the metadata and logs stored by Software Configuration Management (*SCM*) tools or bug tracker tools, which are supported by ontologies in the most advanced systems [10–12].

[1] Source code files, classes and interfaces are usually referred as software items in this research paper. Therefore, Gridmaster represent software items at the file level as well as the class and interface level, whereas the Socio-Technical Graph represent contributions at the file level.

To keep all the members of a team and, in particular, the project administrators informed about what is happening in the development and maintenance of a software system, it is not sufficient merely to rely on the details of the activities that have been carried out recently. It is necessary to perform a detailed analysis, usually for a period of development time and exceptionally for the entire period of development, and provide methods that facilitate comprehension [9].

The aforementioned analysis is known as software evolution analysis, and its principal objectives are to provide information that contributes to the understanding of the Software Evolution (*SE*) process, thus supporting the improvement of system development and maintenance processes, including project management and, as a result, improving human resource management. However, *SE* analysis produces large and complex datasets due to the number of variables involved in the process of change and the complexity of their relationships, which makes an adequate analysis difficult. Although the result of analysing the evolution of software elements provides useful information, it does not provide sufficient information to carry out the tasks of understanding changes and project evolution in a satisfactory fashion to provide adequate support for decision-making. This has led to the following research question being formulated:

> *How can software project leads and managers be supported in the decision-making process with regard to human resource management and the tasks carried out by programmers, by using the analysis results of the evolution of software projects and the socio-technical relations that are established during its evolution?*

Research efforts have focused on the use of visual representations combined with interaction techniques to gain insight when using large and complex datasets. In the context of software systems, these research efforts have concentrated on Software Visualization (*SV*) [13] and Software Evolution Visualization (*SEV*) [14]; although, more recently, some research has been carried out on the application of Visual Analytics (*VA*) to software systems [15] and SE [16] with the aim of providing better results. The aim of such research is to support the process of understanding *SE* and improve the design and implementation strategies of tools directed to satisfy the analysis needs of both programmers and managers.

Therefore, to obtain the results that will answer the formulated research question, it is necessary to design and implement an architecture that demonstrates the usefulness of the application of *VA* to *SE* by using, as data sources, the metadata and source code stored by *SCM* tools for each revision of a software project. However, the application of *Va* to *SE* is relatively new and a definition of this process is lacking. Thus, this makes it necessary to produce a definition that can be used as a base for the design and implementation of the aforementioned architecture.

Consequently, to define the process of applying *VA* to *SE*, it is necessary, on the one hand, to describe the process of applying *VA* to *SE* and, on the other hand, the identification of the roles, borders, interactions and relationships between research areas, methods and techniques involved in such a process.

In accordance with what has already been stated, a secondary objective of this research is to define the process of applying *VA* to the evolution of software systems and answer the following research question:

> *How can the process of applying Visual Analytics to the evolution of software systems be defined and explained, taking into account the components, methods and techniques involved in this process?*

The remainder of this research paper addresses the following topics: Section 2 discusses the background of the application of *VA* to *SE*, Section 3 describes the process of applying *VA* to *SE*; Section 4demonstrates the utility of the process by the definition of Maleku[2], an architecture that was implemented based on the aforementioned process; Section 5explainsthe visualizations and interaction design; Section 6 discusses the use of collaboration patterns and socio-technical relationships for human resource decision-making by means of a case study; Section 7 presents the results of the user assessment test of Maleku, and finally, in Section 7, the conclusions are presented.

2. Background

VA 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, XML files, metadata and source code. It iteratively collects and pre-process data, carries out statistical analysis

[2]The architecture was named after Maleku to honor a small native group from the region of Guatuso, Costa Rica, with an ancient tradition in designing and decorating colorful masks and drums.

[17], knowledge representation [18], user interaction [19], visual representations [20], human cognition [21], perception, exploration and human capacities for decision-making.

Although *VA* is partly based on the use of Information Visualization (*IV*) principles and techniques, its definition overlaps, partially, with that of *IV*: both address data acquisition methods, data transformation, visual representations, human computer interaction and the human capacities for decision-making [22,23]. However, the main difference is that *VA* makes intensive use of data analysis, Coordinated and Multiple Views (*CMV*) [24] and combines the advantages of machines with human strengths, such as analysis, intuition, problem solving and visual perception. Thus, the human factor is a key element for *VA*, and Human Computer Interaction (*HCI*) is a crucial component that supports knowledge discovery.

In particular, *CMV* is concerned with the use of several visualizations that are linked by a model or architecture that coordinates the interactions between them and the data that visualizations must represent, in accordance with the interactions performed [25]. Its use requires a combination of different visualization types (hyperbolic trees, graphs, treemaps, radials, parallel coordinates and grids to name but a few, which are relevant to the present work) to exploit the advantages that each one has to offer [26] and to provide analysts with different levels of detail.

Using *CMV*, analysts can understand relationships among elements located in separate, but linked, visualizations. Additionally, they can explore data from many different viewpoints and have more interaction paths available that may lead to knowledge discovery. Moreover, *CMV* make *VA* tools more scalable, compared to *IV* itself, in terms of data, dimensionality, information complexity and the dynamic feeding of new data [27].

VA has been applied comprehensively to problems as diverse as social networks [28], temporal data analysis [29], visual e-learning analytics [30,31], visual ontology analytics [32,33] and software systems [14]. Moreover, one can say that knowledge discovery is an intrinsic property of *VA*, as it is aimed at supporting analysts and decision makers in gaining insight from large multivariate datasets.

Accordingly, *VA* designs should be centred on the user, attempt to facilitate usability and reduce memory load on users [34]. Its ultimate goal should be to hide complexity details from users and provide an environment for knowledge discovery by means of a fulfilling human experience [35]. Hence, regardless of the complexity of the problem at hand, the success of any *VA* solution depends on the appropriate design of the visual representations and use of interaction techniques.

Taking into account these factors, one of the properties of VA is the ability to provide support for decision-making [36, 37] by using the cognitive abilities of users, and hence, *VA* principles have been applied to software systems [15, 16].

The application of *VA* principles to software systems is known as Visual Software Analytics (*VSA*) [16,38], and it is an improvement relative to *SV*, considering *VA* as a comprehensive process which includes advanced data analysis, the use of multiple linked views and the reasoning abilities of analysts. Therefore, *VSA* has produced promising perspectives in the past few years when applied to software systems and, particularly to topics such as requirements engineering, maintenance, product assessment, system optimization, program structure and metrics comprehension [15].

In this context, it must be highlighted that the tasks performed by project managers and their information needs are complex [6], so this implies a great number of challenges that must be overcome to successfully support project managers in decision-making. Amongst the challenges that remain to be overcome are the following:

1. To facilitate visual analysis of the development process and evaluation.
2. To provide methods to visually monitor the evolution of the quality of software elements (classes, packages, modules and project), taking into consideration the use of software quality metrics with the aim of maintaining complexity and project evolution under control as well as assuring quality control.
3. To provide visual mechanisms to review the measurements of task execution, and permit progress analysis and performance prediction.
4. To assist, using visual methods, the determination process as well as risk management, as well as the size and complexity of the software product.
5. To keep project managers informed on patterns of collaboration between developers and informed of those elements which have been modified either synchronously or asynchronously, as well as the implications (in terms of quality and functionality) of the changes that have been carried out.

The application of VA to SE [9] is a specialization of VSA. A practical analogy is that VSA is like a movie frame while the application of VA to SE is a movie that is composed of multiple movie frames temporally ordered and interrelated. Consequently, one of the advantages of the application of VA to SE is that it takes into account the time dimension; thus, it may offer solutions to overcome some of these

challenges and for the construction of a shared knowledge space for the software development and maintenance processes.

3. The Application of Visual Analytics to Software Evolution

This research defines the process of applying *VA* to *SE* as Evolutionary Visual Software Analytics (*EVSA*), which is a data transformation process that could be thought of as a funnel where raw data are analysed and filtered in several steps until these are converted into knowledge. The output of the process is a reduction, in terms of volume, of the original input, which contains all the required elements to inform decision-making. The EVSA definition takes into account the analytical process proposed by van Wijk[36], the adaptation for *VA* of the *Visual Information - Seeking Mantra*[37], the *IV* model proposed by Card [22], the visualization process proposed by Chi [23] and other previous definitions of this process [38]. Accordingly, the functions and responsibilities of the modules that comprise the *EVSA* process are described in Table 1 and illustrated by Figure 1, and its conceptual definition is the following:

Evolutionary Visual Software Analytics is the process of applying Visual Analytics to Software Evolution with the aim of supporting software development and maintenance, with the active participation of users, through the understanding and comprehension of software change and evolution by means of Visual Analytics and Human-Computer Interaction.

Module	Description
Extract, Transform and Load (*ETL*)	This module has the function of performing connection and data retrieval from software repositories, defect-tracking systems, emails, source code revisions, testing systems, logs and any other available data source. When the data are retrieved, it is cleaned, merged and loaded into a data warehouse.
Advanced Software Evolution Analysis (*ASEA*)	This is comprised of analysis techniques, e.g., [39], that could be used on an individual basis or combined to extract knowledge facts and store them in a database.
Visual Knowledge Explorer for Software Evolution (*VKESE*)	This module is made up of three components: Software Evolution Visualization (*SEV*), Views Linker and Facts Analyzer (*VLFA*), and Visualization Abstractions and Coordination Support (*VACS*).

Table 1: Responsibilities and functions of the modules that make up the Evolutionary Visual Software Analytics process applied to Software Evolution.

The *EVSA* process has been defined using a modular-based approach to facilitate the extension of architectures through the addition of new components, where each module is a collection of components (which are, in turn, formed by methods and techniques). Consequently, three modules constitute the process: *ETL*, *ASEA* and *VKESE*.

ETL is the first module that intervenes in this process and comprises several components aimed at retrieving, cleansing and integrating data from data sources, such as software repositories, defect-tracking systems, emails, source code revisions, testing systems, logs and any other data source. In contrast, the aim of the *ASEA* module is to produce *Knowledge Facts* using, for example, software repository mining, origin analysis, contribution analysis, frequent coupling mining, source code differencing, refactoring analysis, defect classification, architecture and structure analysis, and metrics and code smells detection. *ASEA* carries out intermediate steps in the process of transforming data into knowledge. Its results provide important information that could lead to decision-making, but the results are still unmanageable because of the large volume when dealing with *Big Data*. Thus, the presentation of thousands or even millions of *Knowledge Facts*, as a whole, is not feasible and still requires additional steps for providing usable knowledge that could be successfully employed in informed decisions. This is achieved with the use of *IV* and *HCI*.

The other component of the *EVSA* process is *VKESE*, which consists of three components: *SEV*, *VLFA* and *VACS* (see Table 2).

SEV plays a central role for this module and consists of a set of visualizations. This component makes use of the *VLFA* component to define the associations between *Knowledge Facts* and the visualizations and to define how the visualizations are linked together. It makes use of several theories, methods and techniques, such as usability principles, multidimensional and multivariate visualization,

HCI and information design theories, among others. In turn, *VLFA* carries out an automatic selection of the *Knowledge Facts* that will be visually represented in accordance with the requirements of the visualizations.

Sub-modules of the Visual Knowledge Explorer	**Description**
Software Evolution Visualization (*SEV*)	This component is the most important element of *VKESE* and the *EVSA* process: it provides the visual representations and interaction mechanisms for supporting users in the knowledge discovery process.
Views Linker and Facts Analyzer (*VLFA*)	This should provide an interface for allowing users to create associations between visualizations and *Knowledge Facts*. Moreover, it must support the definition of the linking and coordination of visualizations based on the common attributes of facts (i.e., an approach based on a relational data model [26].
Visualization Abstractions and Coordination Support (*VACS*)	The responsibility assigned to this component is to create the required data models, data structures and visual mappings. It also has the function of coordinating the data to be displayed by the visualizations.

Table 2: Sub-modules of the *VKESE* module.

VKESE also makes use of *VACS* for the creation and management of data models, data structures and visual mappings in addition to keeping track of the interaction and coordination between visualizations for deciding on the data to be visualized according to the interactions and the linking between visualizations.

The steps followed by the *EVSA* process (see Figure 1) are organized into four phases and listed below:

Phase I: Data Retrieval and Loading. It retrieves and carries out an initial data processing, after which it stores data into a data warehouse.

Data retrieval: According to the type of task that the researcher or designer seeks to support, the recovery process can be performed in software repositories, defect-tracking system logs, emails, source code and testing system logs (Load, Arrow 1).

Data warehouse: Once the data have been recovered, it is then cleaned, integrated and correlated and then stored in a data warehouse (Read data, Arrow 2).

Phase II: Data Analysis. This phase analyses and extracts *SE* facts and then proceeds to store the results in a database.

Analysis and facts extraction: When new data are available in the data warehouse, *ETL* reads the data (Read data, Arrow 3) and then *ASEA* proceeds with the analysis, using one or more analytical techniques, depending on the task being undertaken.

INSERT FIGURE 1

Figure 1: Overview of the Evolutionary Visual Software Analytics process

Storage of evolution facts: Once the analysis has been carried out, the evolution facts are then stored in the *Software Evolution Facts* database (Produce facts, Arrow 4).

Phase III: Structure Loading and Visualization Mapping. The tasks of this phase include loading the *SE* facts, creating the data structures and visual mappings, and loading the visualizations.

Visualization loading: The user launches the *SEV* component, which uses linked visualizations. Some of the visualizations that can be used are shown in Figure 1.

Data fact structures request: When the *SEV* component is loaded, the data fact structures required by the visualizations are requested by the *VLFA* component (First analysis, Arrow 5).

Facts loading: The *VLFA* component reads facts from the *Software Evolution Facts Database* and passes them to the *VACS* component (First analysis results, Arrow 7).

Structures and visual mappings: The *VACS* component creates and passes the appropriate data model, data structures and visual mappings to the *SEV* component (Show what is important, Arrow 8).

Phase IV: User Interaction and Details on Demand. This phase is the final stage of the process of transforming data into knowledge. It makes possible a feedback loop between the user and the system: the user requests additional data to the system by means of the interaction possibilities available, and the system provides the requested data. According to user interactions, the knowledge discovery process is refined and progresses towards the finding of useful knowledge and answers.

User interaction: During the process of knowledge discovery, the user browses, filters and explores different perspectives on the data, selecting elements from one or more of the visualizations (Zoom, filter, interact, Arrow 9).

Requesting details: According to the needs and the interactions of the user, the visualization requests new data fact structures and visual mappings to provide additional information to the user in accordance with the options selected (Request of details on demand, Arrow 10).

Additional details: If the additional details that have been requested are available in the form of data, fact structures and visual mappings are passed to the *SEV* component (Details on demand, Arrow 8). However, if these details are not available, a request is passed to the *VLFA* component, which reads the additional facts (Read facts, Arrow 6), transforms the details (Further analysis results, Arrow 7) and then passes them to the *VACS* component so it can proceed to create data fact structures and visual mappings.

Discovery of knowledge: The user continues to interact with the system until the knowledge necessary is obtained or it is considered impossible to reach a determinate conclusion using the data and representations available.

The processes performed by the *ETL* and *ASEA* modules are usually visualized once the execution of these modules has been successfully completed for an initial data volume determined by the analyst. When new data are added to the data sources, the processes performed by these modules should run automatically to generate new *Knowledge Facts*, which are stored in the facts database to automatically update the visualizations.

The application of *VA* to *SE* takes into account the analysis of two or more system revisions which entails carrying out an individual analysis of each revision and then compare and correlate the results in an endeavour to discover relationships, similarities and dissimilarities between these relationships, the following factors should be taken into account:

1. Different types of data are visualized in different time scales (years, months, days and hours), which are also correlated.
2. The visual representation of system structural changes is a difficult endeavour due to the correlation of changes in time.
3. Developers and managers must be much more skilful and cautious in noting relationships and differences when several software project revisions are examined and compared.

Therefore, the application of *VA* to *SE* is similar to a movie that is composed of multiple movie frames that are temporally ordered and interrelated.

4. Maleku: Architecture

Maleku aims to help software practitioners when correlating metrics, project structure, inheritance, interface implementation and socio-technical relationships. The architecture explained in this research has been implemented in Java and tested on several open source software projects, such as *jEdit*, *JabRef*, *ArgoUML*, *Jmol* and *JFreechart*.

INSERT FIGURE 2

Figure 2: Overview of the architecture for Maleku

The modules of the architecture (see Figure 2) are similar to those of the process described in the previous section. The operation of the *ETL* and *ASEA* is synchronous, while the operation of *VKESE* is asynchronous in relation to the other two modules. The architecture is based on the client/server architecture, in which *ETL* and *ASEA* are executed by the server and *VKESE* is an *Eclipse* plugin executed

by the client. The different modules and components of the architecture are described in the following order: data retrieval, data analysis and visual representation. The *ETL* module comprises a sub-module (SM) and two components (C), as follows:

Data Source (C): It consist of the *SCM* software repositories of projects under analysis, from which metadata, programmer's activities, project structure and source code are extracted.

Sensor of New Revisions (C): This process continuously monitors the addition of new revisions to software projects and notifies the *Data Extractor*.

Data Extractor (SM): This sub-module comprises the *Architecture and structure retrieval*, *Source code retrieval*, and the *Metadata retrieval* components.

Architecture and structure retrieval (C): This is responsible for extracting details of the project structure for each revision, with particular interest on the packages of the system and their organization.

Source code retrieval (C): This is responsible for recovering the source code for each of the system revisions and for storing basic information of classes and their location in the system architecture.

Metadata retrieval (C): This component is responsible for retrieving the logs associated with each revision and its details, such as the date on which the revision was carried out, which programmer carried it out and which elements were affected.

The sub-modules that conforms *ASEA* are Source Code Analyzer and Metadata (*SCAM*) and Software Evolution Analysis and Correlation Engine (*SEACE*), the components and descriptions of which are explained below.

Source Code Analyzer (SM): This sub-module is responsible for carrying out the analysis of revisions using the following components:

Metrics detection (C): This component detects and calculates metrics (e.g., *LOC*, *NOM* and *Cyclomatic Complexity*) using details from the parsed source code.

Item relationships analysis (C): This component detects of inheritance (parent-child and child-parent) and interface implementation (implementing and implemented by) relationships.

Source Code Parser (C): This reads each source code file line by line to identify classes, interfaces, methods and declarations, and applies parsing rules. Its use allows for the calculation of metrics and identification of the relationships between software items.

Parsing rules database (C): It applies parsing rules that were generated automatically and manually.

Metadata, software evolution analysis and correlation engine (SM): This is invoked by *Source Code Analyzer* and its function is to identify socio-technical relationships and determine the contributions made by individual programmers, as well as to analyse the architecture and structure of the project for each revision under analysis. The components of this sub-module are:

Contribution analysis (C): Based on the metadata of *SCM* repositories, a cumulative calculation of the elements changed for each revision and programmer is carried out.

Socio-technical analysis (C): Using the metadata of *SCM* repositories as a base, both the relationships between programmers and software items are examined, as well as the relationships that are created between programmers using as a basis the elements that have been changed in common.

Architecture and structure (C): The results produced by *Source Code Analyzer* and the information obtained from the metadata of *SCM* repositories are used to correlate software project structure, metrics and relationships between software items. It further gathers information about the creation of software items and their lifeline during the project.

Software evolution facts data base (C): This database stores the analysis results produced by other sub-modules and components of *ASEA*. To do this, it uses a hierarchy that emulates the software project: project→ revision → package → file → software item.

The sequence of steps that follows the process of data recovery and analysis (made up of *ETL* and *ASEA* modules) are:

1. The user enters the connection parameters of the *SCM* repository and database to be used.
2. Data extractor carry out the task of data recovery (Extract, Arrow 1).
3. Once data have been recovered, it is loaded into the *Data Warehouse* (Load, Arrow 2).

4. When project data have been retrieved, the *Sensor of new revisions* monitors the availability of new revisions in the *SCM* repository and notifies the retrieval modules in a timely manner.
5. As data are retrieved, *ETL* informs *ASEA* that new data are available to perform the respective analysis concordant with the analysis components available.
6. Source code analyser reads the *Data Warehouse* (Read source code, Arrow 3) to detect and calculate metrics and to analyse the relationships between software items.
    6.1 To carry out its tasks, *Source Code Analyzer* requires parsing the source code, and then the Source code parser component is notified. (Callparser, Arrow 4).
    6.2 The component *Source Code Parser* reads the parsing rules from its own database (Read parsing rules, Arrow 5) and performs the parsing of the source code.
7. When *Source Code Analyzer* has finished the analysis, it stores the results in the *Software Evolution Facts Database* and notifies the *Metadata, Software Evolution Analysis and Correlation Engine* (Call second level analysis engine, Arrow 6).
8. *Metadata, Software Evolution Analysis and Correlation Engine* reads evolution facts from the *Software Evolution Facts Database*, as well as metadata, project structure and evolution details from the database of *ETL* (Read metadata, project structure and evolution details, Arrow 7). Using this information, the module then carries out a more profound analysis regarding socio-technical relationships, analysis of the contribution of programmers and the architecture and structure of the software project under consideration.
9. The process that carries out the *ETL* and *ASEA* modules runs indefinitely for each of the projects configured until the analysis for one or more projects is detained by the user.

This architecture permits the addition of new components to modules and sub-modules to allow connections to be made to new data sources performing other types of data analysis and to visualize the results of the analysis with new visual representations. The steps followed by *ASEA* are the same as those that were described in Section 2 with regard to this module; hence, the explanation of these steps is omitted and it is recommended that this section be reviewed for details. In the next section, we proceed to describe the visualizations used by the *SEV* sub-module.

5. Maleku:Visualizations and InteractionDesign

VKESE, the visualization module of *Maleku*, was developed in *Java* as an *Eclipse* plug-in. It makes use of linked views (organized by an overview + details approach), interaction techniques and a colour code for representing programmer contributions. Figure 3 shows the visualizations that it includes: Granular Timeline (a novel visualization displayed at the left bottom corner), *Gridmaster* (a novel visualization located in the right top panel) and *Social-Technical Graph* (STG) (depicted in the right bottom panel).

Accordingly, *Granular Timeline* (*GT*) provides an overview of the project activities, and *Gridmaster*, as well as *STG*, depicts 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 the numbers and arrows in Figure 3) begins with the analysis of the programmers' contribution patterns in the Granular Timeline and the selection, for further analysis in *Gridmaster*, of one or more time units from the radial view. Afterwards, the user can select one of the displayed time units in *Gridmaster* for presenting the associated socio-technical relationships in the corresponding visualization.

The following sections explain the design decisions taken for the visual representations and how they contribute to knowledge discovery and decision-making in the context of software project management. The sample images were obtained from analysing the open source project *jEdit*.

INSERT FIGURE 3

Figure 3: Main view of the Visual Knowledge Explorer for Software Evolution of Maleku

5.1. Granular Timeline

GT uses a modified circular ring chart layout to show an overview of the temporal dimension of a project (Figure 4). Concentric rings demonstrate the time scales that record change events, from coarse (*years*, outer ring) to fine grain (*hours* or finer, innermost ring), and provides an overview + detail view.

GT can be used to represent any type of quantitative data produced over time, given its nature to depict numerical values and statistic results. However, it is utilized in this research to represent statistics based on the commits of revisions and programmer contributions, as is explained below.

Statistics on revisions: The space within chart cells is used to embed different types of visualizations for representing the number of revisions at the level of detail of a particular cell. The *years* ring cells insert bar charts that show the number of revisions for each month. The *months* ring embeds a height plot chart presenting the number of committed revisions per day of the month. The *days* ring shows revisions in each day; hence, this ring has 30 or 31 cells, and the revisions at this level are depicted by bar or squared charts according to user selection (see Figure 5 for the treemap representation). In each cell, a matrix dot plot is drawn in polar (ρ,ϕ) coordinates, where ρ maps the hour at which a revision was created. The innermost *hours* ring shows revisions made each hour and represents the revisions by means of bar or squared charts.

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. Consequently, squared charts were applied because they employ a space filling algorithm, which makes better use of space, and, additionally, permits better highlighting of individual revisions.

Statistics are displayed in the centre of the visualization as the time units are selected. Note that time unit selections begin in the outer ring, which corresponds to years, and follows the sequence *months*→*days*→*hours* (e.g., 2014→ June 2014→ June 27th, 2014→ June 27th, 2014 at 10), as highlighted in Figure 4.

Programmer contributions: The contributions of programmers are represented by the use of colours, where each colour is matched to a particular programmer. In general, the use of colour can permit the acquisition of information related to the statistics of committed revisions and those who have carried out such revisions. Hence, it is easy to extract patterns concerning the programmers who intervened in the development of a project and the job rotation carried out.

INSERT FIGURE 4

Figure 4: Granular Timeline (*GT*) showing statistics of the revisions committed for *jEdit* during 10 years

5.2. Gridmaster

Gridmaster is based on a tree and a grid representation, two structures widely known by programmers because they are common in computer programming (see Figure 5). The tree structure is made up of all the packages and software items that have been added to the project during its evolution (including packages and software items that are no longer part of the current project version), whereas the grid layout is created by the intersection of rows and columns: the packages and software items are placed in the tree structure and associated to rows, and the time units are linked to columns. The use of both the tree and grid permits the correlation of all the software items involved in the evolution process with programmer contributions, the creation of software items and architectural relationship changes, such as metrics and the addition or removal of inheritance and interface implementation.

Accordingly, some of the features of *Gridmaster* permit the extraction of collaboration details at different granularity levels, similar to *GT*. However, *GT* was designed to offer an overview as well as top-down statistics views concerning the revisions of software systems, while *Gridmaster* was designed as a complementary view to correlate the contributions of programmers with software items. Hence, *Gridmaster* depicts the socio-technical relationships between software items and programmers as well as the lifeline of software items for a particular period of time (which is selected from *GT*) or the entire evolution of a system. The representation of these features is carried out by means of colours, as is explained below.

In *Gridmaster*, colours are attributed to programmers, and the area associated to each programmer depends on the number of contributions made (in terms of committed revisions) as well as on the representation that has been chosen from the contextual menu: whether relative or absolute (see Figure 5 for screenshots of these characteristics). 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. Thereafter, 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 assigned to a programmer contribution is different for each time unit. Thus,$\gamma = \varepsilon/\omega$,where$\omega$ is the number of contributions for the time unit under consideration.

Consequently, the purpose of the absolute representation is to depict the lifeline of software items and packages by using an intuitive approach. Figure 5 shows that the activities performed on the package *bsh* of *jEdit* were carried out between 2001 and 2006. It also displays that this package currently is not part of the latest version of the project, which is corroborated when reviewing the structure of *jEdit* on the *Eclipse* workbench. Moreover, in should be noted for Figure 5 that other packages, such as *com*, *gnu*, *macos*and *macosx*, are part of the first level of the project structure (although *macos* and *macosx* were moved into other packages located at a lower hierarchy level; hence, they continue to be part of the project).

INSERT FIGURE 5

Figure 5: Absolute representation of programmers' contributions, showing project structure and lifelines

The relative representation allows a comparison with precision of the activity carried out in packages as well as the time period that concentrates on more activities and, at the same time, provides visual information regarding a programmer's contribution, as the area assigned to a contribution is the same for all time units in the visualization. Figure 6 shows that the package with the most activities is *org*, 2008 is the year in which most activities were carried out and the programmer with the most activity during the period from 2001 to 2005 is the one represented by the user *spestov* (represented by the green colour and whose name is obtained from the tooltip that is displayed when the mouse is moved over the coloured areas).

INSERT FIGURE 6

Figure 6: Relative representation of programmers' contributions, using 2008 as the reference year

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 the associated months. Moreover, the user has the possibility to choose how metrics and programmers are represented by selecting between relative or absolute value representations.

5.2.1. Socio-TechnicalGraph

STG is displayed when a time unit (e.g., a year or month) is selected from *Gridmaster*. In this scheme, *STG* 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 Figure 7). Edges connecting programmers depict relationships among these programmers based on the changes made to software items. The thickness of edges represents the number of common software items that the associated programmers have changed. Accordingly, *kpouer* (orange) is the programmer that has made more contributions in the selected year (2010), followed by *ezust* (pink), *k_satoda* (light yellow) and *daleanson* (light grey). Additionally, *kpouer* has changed several software items, together with *ezust*, *k_satoda* and *daleanson*. Hence, the relationship between *kpouer* and *ezust* is very close, as can be deduced from the large number of software items on which they have collaborated. Thus, this visualization is useful in scenarios where a programmer's tasks need to be redistributed due to an unexpected situation ororganizational change. Consequently, programming tasks performed by *kpouer* may eventually be taken over by *ezust, k_satoda and daleanson*.

INSERT FIGURE 7

Figure 7: Socio-Technical Graph (*STG*)

Consequently, the size of nodes in *STG* conveys the number of contributions made by programmers, which are determined by the number of files that have been modified in each commit (the calculus of node weights differs from the one made by Jermakovics et al. [40],which only takes into

account the number of commits). The contributions weight, *w*, is the sum of the number of files committed per programmer and per commit and is calculated as follows:

$\sum_{i=1}^{c} F_i$, where c is the number of commits made by the programmer and F_i is the number of files for the commit_i.

The thickness of edges represents the number of common software items that the linked programmers have changed. Hence, the strength of the collaboration relationship between two programmers can be deduced from the thickness of edges. This visualization is useful in scenarios where a programmer's tasks need to be redistributed due to an unexpected situation or organizational change, where, hypothetically, programming tasks performed by programmer ***A*** may eventually be taken over by programmer ***B***.

6. Case Study: Collaboration Patterns and Socio-Technical Relationships for Human Resource Decision-making

This section is aimed at demonstrating the utility of *Maleku* and *VKESE* in knowledge discovery for supporting software project leads and managers in decision-making, with regards to human resources under their responsibility. It is important to highlight that *VKESE* is a client side module that can be used from different locations by several project managers and whose components permit knowledge discovery for several software evolution facts, such as the lifeline of software items, programmers' contributions and socio-technical relationships.

Accordingly, this section focuses on knowledge discovery from socio-technical relationships and looks to answer questions regarding the contributions made by programmers (i.e., Who has led the development of the software project or has contributed the most?), the contribution patterns of programmers (i.e., Why has the programmer made so many contributions in such a short time?), the relationship between software items and programmers (i.e., Who has modified a given software item?),and the software items programmers have changed in common (i.e., Which software items have programmers changed in common?). The discussion in this section will be centred on the aforementioned questions and the analysis of *JabRef*, an open source software project written in *Java* for bibliographic management. However, the questions will be addressed using this project as an example and not considering its open source nature.

GT, as we have already mentioned, represents the contributions of programmers using bar charts (see Figure 4) or a combination of bar charts and treemaps (see Figure 8). 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 because they employ a space filling algorithm, which makes better use of space, and, additionally, permit better highlighting of individual revisions.

Analysing the contribution patterns is an intuitive task with *GT*. Figure 8 shows the contributions made to *JabRef*during 9 years. The use of colour is an important feature in this representation, and the area covered in the visualization by a given colour is proportional to the number of contributions made by a given programmer. Accordingly, it is easy to recognize, at first glance, that the programmer *mortenalver (light purple)* has been leading the project and has made most of the contributions to the project in terms of revisions (note that the names of programmers are obtained from a tooltip that appears when the mouse moves over the corresponding graphic area).

Other programmers made contributions for a time and then left the project; for example, *jzieren* (brown) and *kiar* (dark yellow) both made contributions, but for only a few months in the years 2004, 2005 and 2006. Other programmers with important contributions are *coezberk* (purple) and *olly98* (light green).

Another relevant feature of this visualization is its capability for uncovering contribution patterns. Figure 8 makes evident that *jzieren* (brown) made revisions several times an hour, as can be noted from the innermost ring (the 'hours' ring) on the days with activity associated to this programmer. This can lead to many questions from a software project lead or manager, among which are: Why has this programmer made so many commits? Has this programmer corrected errors on previous commits, then fixed them and commit again? Is this programmer accustomed to making commits as an analogy to saving changes to the project? Moreover, this visualization permits software project leads and managers to obtain insight contribution patterns associated to the working hours of programmers: one can select a given day and review the hours in which programmers have made commits, revealing the working pattern of developers. The answer to the previous questions and the contribution patterns could help the software

project manager to make decisions regarding the compliance of human resources to rules and training. A relevant decision for software project leads and managers, after this analysis, is the one concerned with the programmers that require training and the contents of such training.

Contribution patterns could be observed in *GT* as well as in the *Gridmaster*, where colours are also used to show the volume of contributions made by programmers for one or more time units. However, *Gridmaster* also correlates contributions with the project structure down to the file level, providing details of the software items that have been changed by each programmer. The aim of this feature is to assist software project leads and managers in the assignment of programming tasks to programmers according to their previous experience based on the changes made to software items.

INSERT FIGURE 8

Figure 8: Granular Timeline: frequency of contributions of *jzieren* for April 20, 2005.

In line with this analysis, Figure 9 shows that *mortenalver (light purple)* has made changes to all the packages in the project, except the package *antlr*, which was created and changed only by *jzieren*(brown). Moreover, *coezbek* (purple) also contributed to most packages in the project during the years 2006, 2007, 2008 and 2009; hence, he could have substituted for *mortenalver* in the case of an eventuality. A similar pattern can be observed in 2011, when*olly98* (light green) contributed significantly to some important packages and, therefore, could take over some tasks assigned to *mortenalver*. The packages modified by *olly98* are *net*, *spl* and *tests*, of which *net* is the core package of the system and *tests* is the package associated to the system's tests cases.

INSERT FIGURE 9

Figure 9: Gridmaster view for the correlation of project structure, time and contributions of programmers.

Thereafter, *STG* reveals the programmers that have made more contributions to the project and the relationships between them, using as reference the software items they have changed in common, as shown in Figure 10. The larger node represents *mortenalver* (light purple), the programmer who has made more contributions to the project; other programmers with important contributions are *coezbek* (purple), *jzieren* (brown) and *olly98* (light green). The edge between nodes depicts a relationship between programmers, and the thickness of the edge is proportional to the number of software items that both programmers have changed in common.

INSERT FIGURE 10

Figure 10: Visualization of the contributions and relationships between programmers for *JabRef*

The advantage of *STG* is its capability to represent, in a small screen area, several years of data, as shown in Figure 10, where 9 years of information has been summarized. While this visualization represents the complete evolution period, it is valuable for understanding the contributions made to the project and the interactions between programmers. Hence, the software project manager requires a representation of a recent and shorter time period for deciding on programmers' substitutions and the collaboration relationships between programmers, which can be carried out by choosing a particular year from Gridmaster to obtain the corresponding depiction with *STG*.

7. UserAssessment Test

The design of Maleku takes into account the three components of the Human Performance Model [41] in order to achieve the functionality and usability required to perform the tasks it aims to support. Maleku is targeted to programmers and project managers (the human) who are engaged in tasks related to software development and maintenance (the activity) within a software development department or a software company (the context).

The purpose of this section is to assess VKESE using an assessment test to verify its design, functionality and usability. The selection of this type of test is based on its characteristics to evaluate tools in intermediate development stages, as it is the current state of Maleku. It should be noted that the evidence presented in this section is part of an iterative approach to design, implement and test tools, so that the results presented here are part of the first iteration of this iterative process.

The objective of this evaluation is to verify the degree of satisfaction and fulfilment of user expectations [42], based on the goals and tasks that the tool is intended to support. Accordingly, this test seeks to confirm that VKESE:

1. Offers statistical information about the revisions performed during the development of a software system.
2. Makes it possible to determine the contributions made by programmers using as a basis their commits (revisions).
3. Permits to identify the lifelines of software items (including packages, files, classes and interfaces) and their relationship with programmers.
4. Provides details on the collaboration between developers in source code programming.

Accordingly, the following is the research question of the user assessment test:

Do the design and the integrated use of the visualizations in VKESE and their results satisfy users' expectations regarding usability and the support offered to decision-making with regard to human resource management during the software development and maintenance processes?

Position	Industry	Age	Degree	Experience		
				Professional		VA
				Current position	Total	
Programmer	Financing	37	Master	15	15	0
		32	Master	1	7	5
	Software company	38	Ph.D.	1	15	0.5
Research and development	University	35	Ph.D.	2	8	6
	Software company	34	Ph.D.	1	12	5
Architect	Mining	41	Master	9	19	0.5
Team leader	Financing	39	Master	7	16	0
Average		**37**		**5.14**	**13.14**	**2.29**

Table 3: Background details of the participants in the usability study.

The steps followed for the preparation of the assessment test were the identification of users and tasks (see Table 3), the definition of the tasks to be performed by users and the setup of the test environment. The tasks that users were asked to undertake during the assessment tests are the following:

1. Explore the tool for 15 minutes to familiarize themselves with its general operation and the interaction between views.
2. Explore the activities carried out by programmers to understand their contributions and the socio-technical relationships that they have established with the software items. Participants were also asked to obtain details about who has led the project under study and which programmers are actively participating or have ceased its participation in the development of the system.
3. Carry out an assessment of the visualizations, based on the type of representation used and their ease of understanding and learning.

Concerning the setup of the test environment, it was carried out on an individual basis, because the users were located in different countries and cities. To this end, users needed to use software for video conferencing as well as remote access software for accessing a server, where the tool was running.

7.1. Assessment of the comprehension of the collaboration among programmers

Users were asked to perform a number of tasks with the use of the visualizations and then, they answered 7 questions. This group of questions was targeted to aid the comprehension of the relationship

among programmers and to support managers and team leaders in decision making. Questions Q3, Q4, Q5 and Q6 were answered correctly by all participants, whereas questions Q1, Q2 and Q7 were answered incorrectly. It is noteworthy that to answer questions Q1, Q2 and Q7 only one visualization was required, and not several, which contrasts with the other questions that required the use of more than one visualization. In addition, it is also interesting that these questions are related to the contributions of programmers and their continuity in the project.

The participants who answered incorrectly questions Q1, Q2 and Q7 showed confusion because they did not understand which visualization should be used to answer the questions. Their feedback points out that there were no explicit indications in the screen about which visualization to use for accomplishing each particular task, neither the steps that could be followed. This could indicate that the design of VKESE is not intuitive enough and that its learning requires training to understand better the visualizations that compose it (the training that was offered to participants was of only 15 minutes).

The average of correct answers for this group of questions was 5:86 out of 7 (83:67). The questions and the results for this group are listed in Table 4. The average time required by each user to answer these questions was 25 minutes on average, 3:57 minutes per question.

	Question	Results
Q1.	Who is the programmer that has created more revisions during the project evolution?	4
Q2.	Who is the programmer that led the software project in its early stages?	5
Q3.	Select a time period in Gridmaster, see the results in the STG view and correlate the information with that shown in Gridmaster. Then, based on the software items that programmers have changed in common, describe the relationship among them.	7
Q4.	Select the year 2012 in the Gridmaster and review the results in the Socio-Technical view. Based on the results, who is the programmer with more contributions in 2012?	7
Q5.	Based on the volume of revisions, who are the programmers that have taken most of the project responsibility during the year 2012?	7
Q6.	Based on the number of revisions in 2012, who are the programmers that could substitute the leading programmer in any eventuality?	7
Q7.	Who are the programmers that could have left the project in the past year?	4

Table 4: Question group: Collaboration among programmers.

7.2. Evaluation of the tool

The goal of the questions in this section is to assess VKESE, in general. This group of questions was asked after the users had used the tool, looked for details and answered the group of questions of the previous section. It is composed by closed-ended and open-ended questions that are aimed to evaluate the visualization design of the tool in general. Closed-ended questions assess the visual design, ease to learn and user satisfaction with regard to the visualization, whereas the entry values provided to open-ended questions are shown as edited comments. Close-ended questions are graded between 1 and 5 (see Table 5) and the results are presented as an average value of the answers provided.

Weights	Answers	
	Evaluation	User satisfaction
1	Strongly disagree	Not at all satisfied
2	Disagree	Not satisfied
3	Neither agree or disagree	Partially satisfied
4	Agree	Satisfied
5	Strongly agree	Highly satisfied

Table 5: Options for closed-ended questions to assess the visual design, easy to learn and user satisfaction.

The results of the questions in this group (9 questions) are presented in Table 6, where questions Q8, Q9 and Q10 are closed-ended questions and questions Q11, Q12, Q13 and Q14 are open-ended questions. Question Q8 assess the visual design, question Q9 how easy to learn, use and understand is the tool and Q10assess the satisfaction degree. By contrasting the rating given to question Q8 with the answers to the questions listed in Table 4, it can be noted that there is a significant discrepancy between the answers.

The reason for this discrepancy could be a misinterpretation of this question, due to conceptual issues, because multiple users did not understand the notion of the term socio-technical clearly enough.

The rating obtained by question Q9 is not positive when a correlation with the comments offered to question Q13 is made: participants suggested to improve several aspects regarding the ease to learn of VKESE. The average rate for questions Q8, Q9 and Q10 was 4 out of 5 (80%), as listed in Table 6.

The answers provided for question Q11 are shown in Figure 11 and revealed that the main concerns of participants for adopting these kind of tools are User overload, Ambiguity, Over-complexity, Difficulty to understand, training and resources demands and the requirement of previous knowledge. With regard to questions Q12, Q13 and Q14, the text entry comments provided are the following:

> **Question 12:** The integration of the tool as an IDE plugin is one of the strongest points of this toolset, because it allows to have everything in the same programming environment. However, the performance of the tool needs to be improved. The tool allows to analyse intuitively the life cycle of a project and it could be useful for software maintenance. The tool is simple, easy to learn and allows to represent large datasets cleanly.
>
> **Question 13:** The interaction between the visualizations needs to be improved, as well as the performance of the tool. Learning the features of the tool requires some time, as it is a specialized tool. Furthermore, the volume of details displayed to the users could easily overload them. The visualizations also lack from customizable features.
>
> **Question 14:** The tool requires to improve the performance of data loading.

	Question	Question type	Results
Q8.	Is the tool clear enough to provide insight into the socio-technical relationships among programmers and software items?	Design and learning	3.85
Q9.	Is the tool easy to learn, use and understand?	Design and learning	3.71
Q10.	What is your satisfaction degree with this tool?	Design and learning	4.14
Q11.	Which were disadvantages of the tool evaluated?	See Figure 11	
Q12.	Which are positive aspects of this visualization?	Open-ended	
Q13.	Which are negative aspects of this visualization?	Open-ended	
Q14.	Please add any extra comments you have.	Open-ended	

Table 6: Global assessment of VSEKE.

INSERT FIGURE 11

Figure 11: Blockers for the adoption of the tool.

7.3. Discussion

Comprehension of the collaboration among programmers: The results for this evaluation were positive: the average grading for the tasks in this group was 83:67%.However, it is worth to highlight that questions Q1, Q2 and Q7, which are related to the contributions of programmers, obtained poor ratings. Thus, the aspects of VKESE to be improved should be reviewed carefully, so users can obtain information more efficiently. However, overall, the answers of participants show that the tool meets the functionality goals that are pursued with its design.

Tool evaluation and visualization design: The goal of the assessment of the visualization design was to measure the satisfaction of users when using the tool for finding relevant details to accomplish a given tasks. In this context, some closed-ended questions were used to gather information on the overall satisfaction of users and open-ended questions were used to get further information to improve the functionality and usability of the tool. The results obtained from closed-ended questions in the usability group, in general, were positive, as the average answer was 4 out 5 (80%).

Q11 received the lowest rating in the evaluation of the tool, with regards to the satisfaction degree of participants with the visualization design. This rating appears to be related to the ease of use and learn of

the tool, the answer to question Q10 reflects the second lowest rating in the assessment. Therefore, although participants considered that information is adequately represented, this does not imply that it was easy for them to use the visualizations and extract useful knowledge.

Finally, some comments from users pointed out on the improvement of the tool performance, the addition of searching and navigation features, the need for additional functionality to get more project details and the inclusion of custom features such as the ability to customize colour coding. Another detail to be considered is that the interaction options are not clear to users, so it is advisable to develop strategies to make more intuitive the interaction possibilities of the visualizations. Moreover, the evaluation also allowed to detect some minor bugs that were not previously detected.

According to these results, VKESE meets the goals and objectives that were raised during its design to support managers and team leaders. Thus, the visualizations that constitute this tool represent and provide relevant information in an appropriate form, although the degree of satisfaction and fulfilment of expectations of users when using them is positive it is not optimal and requires attention in the next iteration of the design and implementation process.

8. Conclusions

Decision-making concerning human resources is based on the events and activities performed in the day to day activities of the production environment and is frequently a shared responsibility between the human resources department and managers, where the feedback and decisions of software project leads and managers are of great importance. Therefore, software project leads and managers must monitor and control tasks with the goal of using collected information to facilitate decision-making regarding personnel. However, the high turnover of personnel between companies and projects within the same company affects decision-making because software project leads and managers frequently make decisions empirically and without the use of tools, which is a drawback when someone is new to a project or does not have enough available information to carry out an analysis. In this regard, the current research has not found evidence of the existence of tools, which make use of *SCM* logs and the source code associated to revisions, to support software project leads and managers in the management of human resources. However, this research has made some progress in this direction.

The definition of the *EVSA* process, its detailed description and explanation that may be used for the design and implementation of tools to gain insight on the activities carried out by programmers and to support project managers in the management of human resources. This became evident after designing and implementing the architecture, based on this process and explained previously, which offers a shared knowledge space that intensively uses visualization and interaction techniques. The resulting toolset, and the visualizations it includes, has taken into account several items one needs to be informed about concerning the evolution of software projects. Some pertain to which programmers are participating in the development of specific parts of a project, while others concern when the project has taken place and when the solution becomes stable. Moreover, the visualizations can also inform the software project manager about which programmer has created more revisions and new files, as well as how developers collaborate and work when separated from one another. Hence, the visualizations have contributed to the understanding of the collaboration patterns of programmers and the contributions made by each of them, and hence it can support project managers in the management of human resources. Hence, for example, it has been determined that the project leader of *JabRef* is the programmer who corresponds to the user *mortenalver* (light purple), and the most suitable programmer to assume the majority of tasks assigned to that programmer, taking into account the contributions carried out in 2011, is the one associated with the user *olly98*(light green).In summary, the use of the visualizations, as was explained in Section 5, has demonstrated the utility of the toolset by aiding in answering questions regarding who has led the development of the software project or has contributed the most, who has modified a given software item and which software items have programmers changed in common.

It is also important to mention that the tests conducted using these novel visualizations have only required a few interactions with the visual representations and, thus, have offered very little time to comprehend the represented data. Hence, the importance of fulfilling usability criteria and providing information in an intuitive way takes advantage of a user's cognitive abilities. Accordingly, it can be deduced that tools based on the EVSA process can provide effective methods for achieving knowledge discovery, which supports decision-making in human resources management tasks.

Moreover, *VA* could benefit from the definition of the *EVSA* process, taking into account that the latter is a specialization of the former and, therefore, shares many similarities. Consequently, tool designers of human resources systems can also take advantage of this process definition to design general purpose visual analytics tools for decision makers in human resources departments.

EVSA can help to overcome some of the challenges listed before to successfully support project managers in decision-making. Furthermore, it can also help programmers in the comprehension and

understanding of software evolution and the changes that provide insight into, to name but a few, software project structures, coupling, inheritance and interface implementation, as well as changes in software quality metrics and code cloning. In line with this, future research will address more broadly the use of *VA* to thoroughly support software project leads, managers and programmers according to the observations made.

Finally, the implementation of the architecture (as a proof of concept) based on the EVSA process has some limitations. It was implemented as a desktop application, does not allow annotating events or relationships in similar fashion as *Storyboards* permits and the use of metrics is simple. Therefore, it is open for future work to design and implement web version of *Maleku* as a plugin of a cloud based IDE, as well as allowing managers to annotate events and relationships across the views. Furthermore, it is important to note that the goal of representing metrics in *Maleku* is to shed some light about how the quality of the system has evolved, and therefore, how the contributions of programmers have affected the quality of the system. However, the metrics that are currently represented are simple (e.g., LOC and NOM) and a future improvement of the design is to represent more complex metrics such as cyclomatic complexity to help determine how the contribution of programmers have affected the complexity of the system, and thus, its maintainability.

Acknowledgement

This work was supported by the Ministry of Science and Technology (MICITT) of Costa Rica.

References

[1] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, F.J. García-Peñalvo, E. Tovar, Project managers in global software development teams: a study of the effects on productivity and performance, Softw. Qual. J. 22 (2014) 3–19. doi:10.1007/s11219-012-9191-x.

[2] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, S. Misra, F.J. García-Peñalvo, Analyzing human resource management practices within the GSD context, J. Glob. Inf. Technol. Manag. 15 (2012) 30–54.

[3] R. Colomo-Palacios, P. Soto-Acosta, F.J. García-Peñalvo, Á. García-Crespo, A Study of the Impact of Global Software Development in Packaged Software Release Planning, J. Univers. Comput. Sci. 18 (2012) 2646–2668.

[4] I. Omoronyia, J. Ferguson, M. Roper, M. Wood, A review of awareness in distributed collaborative software engineering, Softw. Pract. Exp. 40 (2010) 1107–1133. doi:10.1002/spe.1005.

[5] S. Misra, R. Colomo-Palacios, T. Pusatli, P. Soto-Acosta, A discussion on the role of people in global software development, Teh. Vjesn. 20 (2013) 525–531.

[6] J. Münch, J. Heidrich, Software project control centers: concepts and approaches, J. Syst. Softw. 70 (2004) 3–19. doi:10.1016/S0164-1212(02)00138-3.

[7] S.U. Khan, M.I. Azeem, Intercultural challenges in offshore software development outsourcing relationships: an exploratory study using a systematic literature review, Softw. IET. 8 (2014) 161–173.

[8] C. Casado-Lumbreras, R. Colomo-Palacios, F.N. Ogwueleka, S. Misra, Software Development Outsourcing: Challenges and Opportunities in Nigeria, J. Glob. Inf. Technol. Manag. 17 (2014) 267–282. doi:10.1080/1097198X.2014.978626.

[9] A. González-Torres, Evolutionary Visual Software Analytics, University of Salamanca, 2015.

[10] G. Beydoun, G. Low, F. García-Sánchez, R. Valencia-García, R. Martínez-Béjar, Identification of ontologies to support information systems development, Inf. Syst. 46 (2014) 45–60. doi:10.1016/j.is.2014.05.002.

[11] M.Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, J.J. Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, Knowl.-Based Syst. 56 (2014) 15–25. doi:10.1016/j.knosys.2013.10.006.

[12] M.Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, J.J. Samper-Zapater, Creating a semantically-enhanced cloud services environment through ontology evolution, Future Gener. Comput. Syst. 32 (2014) 295–306. doi:10.1016/j.future.2013.08.003.

[13] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
[14] A. Gonzalez, R. Theron, A. Telea, F.J. Garcia, Combined Visualization of Structural and Metric Information for Software Evolution Analysis, in: Proc. Jt. Int. Annu. ERCIM Workshop Princ. Softw. Evol. IWPSE Softw. Evol. Evol Workshop, ACM, New York, NY, USA, 2009: pp. 25–30. doi:10.1145/1595808.1595815.
[15] A. Telea, L. Voinea, Visual software analytics for the build optimization of large-scale software systems, Comput. Stat. 26 (2011) 635–654. doi:10.1007/s00180-011-0248-2.
[16] A. Gonzalez-Torres, R. Theron, F.J. Garcia-Penalvo, M. Wermelinger, Y. Yu, Maleku: An evolutionary visual software analysis tool for providing insights into software evolution, in: 2011 27th IEEE Int. Conf. Softw. Maint. ICSM, 2011: pp. 594–597. doi:10.1109/ICSM.2011.6080838.
[17] R. Peck, C. Olsen, J.L. Devore, Introduction to Statistics and Data Analysis, 3 edition, Cengage Learning, Australia; Belmont, CA, 2008.
[18] F. van Harmelen, V. Lifschitz, B. Porter, Handbook of Knowledge Representation, Elsevier, 2008.
[19] N. Mahyar, M. Tory, Supporting Communication and Coordination in Collaborative Sensemaking, IEEE Trans. Vis. Comput. Graph. 20 (2014) 1633–1642. doi:10.1109/TVCG.2014.2346573.
[20] Y.K. Leung, M.D. Apperley, A Review and Taxonomy of Distortion-oriented Presentation Techniques, ACM Trans Comput-Hum Interact. 1 (1994) 126–160. doi:10.1145/180171.180173.
[21] A. Drigas, L. Koukianakis, Y. Papagerasimou, Towards an ICT-based psychology: E-psychology, Comput. Hum. Behav. 27 (2011) 1416–1423.
[22] S.K. Card, J.D. Mackinlay, B. Shneiderman, Readings in Information Visualization: Using Vision to Think, Morgan Kaufmann, 1999.
[23] E.H. Chi, A taxonomy of visualization techniques using the data state reference model, in: IEEE Symp. Inf. Vis. 2000 InfoVis 2000, 2000: pp. 69–75. doi:10.1109/INFVIS.2000.885092.
[24] N. Boukhelifa, P.J. Rodgers, A Model and Software System for Coordinated and Multiple Views in Exploratory Visualization, Inf. Vis. 2 (2003) 258–269. doi:10.1057/palgrave.ivs.9500057.
[25] J.C. Roberts, State of the Art: Coordinated Multiple Views in Exploratory Visualization, in: Fifth Int. Conf. Coord. Mult. Views Explor. Vis. 2007 CMV 07, 2007: pp. 61–71. doi:10.1109/CMV.2007.20.
[26] C. North, B. Shneiderman, Snap-together visualization: can users construct and operate coordinated visualizations?, Int. J. Hum.-Comput. Stud. 53 (2000) 715–739.
[27] G. Andrienko, N. Andrienko, Coordinated Multiple Views: a Critical View, in: Fifth Int. Conf. Coord. Mult. Views Explor. Vis. 2007 CMV 07, 2007: pp. 72–74. doi:10.1109/CMV.2007.4.
[28] A. Perer, I. Guy, E. Uziel, I. Ronen, M. Jacovi, The Longitudinal Use of SaNDVis: Visual Social Network Analytics in the Enterprise, IEEE Trans. Vis. Comput. Graph. 19 (2013) 1095–1108. doi:10.1109/TVCG.2012.322.
[29] D. Gotz, H. Stavropoulos, DecisionFlow: Visual Analytics for High-Dimensional Temporal Event Sequence Data, IEEE Trans. Vis. Comput. Graph. 99 (2014) 1. doi:10.1109/TVCG.2014.2346682.
[30] D.A.G. Gómez-Aguilar, R. Therón, F.J. García-Peñalvo, Semantic Spiral Timelines Used as Support for e-Learning., J Ucs. 15 (2009) 1526–1545.
[31] D.-A. Gómez-Aguilar, F.-J. García-Peñalvo, R. Therón, Analítica Visual en e-learning, El Prof. Inf. 23 (2014) 236–245.
[32] F.J. García-Peñalvo, R. Colomo-Palacios, J. García, R. Therón, Towards an ontology modeling tool. A validation in software engineering scenarios, Expert Syst. Appl. 39 (2012) 11468–11478. doi:10.1016/j.eswa.2012.04.009.
[33] J. García, F.J. García-Peñalvo, R. Therón, P. Ordóñez de Pablos, Usability Evaluation of a Visual Modelling Tool for OWL Ontologies., J UCS. 17 (2011) 1299–1313.
[34] N. Hollender, C. Hofmann, M. Deneke, B. Schmitz, Integrating cognitive load theory and concepts of human–computer interaction, Comput. Hum. Behav. 26 (2010) 1278–1288.

[35] J. Takatalo, G. Nyman, L. Laaksonen, Components of human experience in virtual environments, Comput. Hum. Behav. 24 (2008) 1–15.
[36] J.J. van Wijk, The value of visualization, in: IEEE Vis. 2005 VIS 05, 2005: pp. 79–86. doi:10.1109/VISUAL.2005.1532781.
[37] B. Shneiderman, The eyes have it: a task by data type taxonomy for information visualizations, in: IEEE Symp. Vis. Lang. 1996 Proc., 1996: pp. 336–343. doi:10.1109/VL.1996.545307.
[38] A. González-Torres, F.J. García-Peñalvo, R. Therón, Human–computer interaction in evolutionary visual software analytics, Comput. Hum. Behav. 29 (2013) 486–495.
[39] H. Kagdi, M.L. Collard, J.I. Maletic, A survey and taxonomy of approaches for mining software repositories in the context of software evolution, J. Softw. Maint. Evol. Res. Pract. 19 (2007) 77–131. doi:10.1002/smr.344.
[40] A. Jermakovics, A. Sillitti, G. Succi, Mining and Visualizing Developer Networks from Version Control Systems, in: Proc. 4th Int. Workshop Coop. Hum. Asp. Softw. Eng., ACM, New York, NY, USA, 2011: pp. 24–31. doi:10.1145/1984642.1984647.
[41] R.W. Bailey, Human performance engineering: Using human factors/ergonomics to achieve computer system usability, 2nd ed., Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
[42] J. Preece, H. Sharp, Y. Rogers, Interaction Design:beyond human-computer interaction, John Wiley & Sons, 2015.