The Small Project Observatory

Mircea Lungu and Michele Lanza
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

Abstract

Maintenance is an important activity in software engineering with studies attributing it more than 75% of the total cost of a system. More than half the time dedicated to maintenance is spent on reverse engineering the code which often is the only accurate source of information. In this context, among the many approaches, software visualization tools have long been seen as an important asset to support the comprehension process, offering facilities to understand a given software system.

However, systems do not exist in isolation, but they exist in the context of organizations, research groups or open-source communities. We call these larger contexts, that contain version repositories for multiple systems in parallel, "super-repositories". We argue that there is a need for software visualization tools that support the reverse-engineering of such super-repositories. We present a tool called the Small Project Observatory (SPO) that supports the visualization and interactive exploration of super-repositories.

Key words: reverse engineering, visualization, tools, super-repositories

1. Introduction

Change is fundamental to software. According to Lehman (18), this is the result of the software development process being a multi-input, multi-output process involving feedback at many levels (18). Parnas showed that the changes which are not done in concordance with the initial design of the system - either because the programmers are not familiar with the code base or due to "*ignorant surgery*" - will result in a system which is harder to maintain. He called this phenomenon *software aging* (27). In a more recent study on a large telecommunications system, Eick et al. showed that as the initial architecture of the system degrades, code quality decreases too, the modularity of the

Email addresses: mircea.lungu@lu.unisi.ch (Mircea Lungu), michele.lanza@unisi.ch (Michele Lanza).

code becomes less crisp and maintenance becomes increasingly difficult and expensive (8). Alan Perills epigrammed this: "In the long run every program becomes rococo - then rubble"(28).

In this context, a very important part of the software engineering process is dedicated to software maintenance. Studies by Sommerville (29) and Davis (5) attribute maintenance between 50% and 75% of the total engineering effort spent in developing a system. A very large part of maintenance is dedicated to reverse engineering the code. Reverse engineering as defined by Chikofski (1) is the process of examination of the subject system in order to

– identify the system's components and their interrelationships and
– create representations of the system in another form at a higher level of abstraction.

The research community has developed over the years a rich set of techniques and tools that support reverse engineering such as clustering, semantic analysis, slicing, visualization, etc. We focus on software visualization, which has a long history in the context of program understanding with research dating back to the mid-eighties.

Software visualization tools span a wide range of abstraction levels. One of the first visualization tools was Eick's Seesoft (7) which summarizes changes in the software at the level of lines of code. Increasing the level of abstraction, Lanza's CodeCrawler used metrics to provide insights into the quality of the systems (15). Girba studied the evolution of whole class hierarchies (12). Some environments focused on visualizing even higher-level abstractions such as Rigi (25) and SHriMP (30) or our previous work on Softwarenaut (21). Holt visualized architectural differences between two versions of a system (14).

To our knowledge, nobody went to the next level of abstraction to visualize the evolution of a software system in the larger context of other systems that are being developed by an organization in an interactive way. In this article we present a tool for the visualization and exploration of project ecosystems, which are groups of projects that are developed together in the context of an organization.

We wanted also to experiment with providing an online software visualization application. There are several other examples of using the Web for software engineering. Holt et al.(9) have developed the Software Bookshelf, which is a web-based paradigm for the presentation and navigation of information representing large software systems. Mancoridis et al.presented REportal which was aimed to be an online reverse engineering portal (23). Recently D'Ambros et al.proposed an online framework for analysis and visualization in a software maintenance context. Their tool, Churrasco focuses on flexibility by allowing easy model extension as well as collaboration (4). Although they support loading multiple models at the same time in churrasco, they do not consider a super-repository as a first-class entity in their analysis. Nentwich presented BOX, an online tool that enables software engineers to access and review UML models in the browser (26). Lin et al.presented an interactive and customizable graph visualization engine implemented entirely in SVG and ECMAScript (19).

2. Super-Repositories

Software systems do not exist by themselves, but they are rather developed in greater contexts such as the software in an organization, a research group or an open-source community. Technically such a context manifests itself in the form of a "super-repository".

We define a super-repository as "physical or logical collection of version repositories for projects that are developed in the context of an organization, a research group or an open source community". A super-repository can be either physical or logical. Based on the versioning system that is being used and the organization's policy, the projects of an organization can be stored together in a single physical repository, such as it is the case with VisualWork's Store, Microsoft's SourceSafe, or IBM's ClearCase. In the same time, all the projects in the RubyForge.org repository are not necessarily stored in the same physical space, but they represent the asset of a single open-source community.

The organization can have multiple forms. In our experience we have seen super-repositories in research groups, in companies and in the open-source world. Evidently, the various types of organizations are very different but characterizing the organization is exactly one of the features of super-repository analysis. We have shown elsewhere how one can compare various organizations in terms of their collaboration structures based on the analysis of their super-repositories (20; 22).

Super-repository analysis is inherently time-dependent. We assume that for every project we have a version repository that stores the evolution of that project. This allows us to analyze both the way every project in the organization evolved as well as other types of information, *e.g.,* which developer worked on which projects at which time, to what extent and in what form did developers collaborate *etc.* This added information makes it important for a company to understand what its super-repository contains and how it evolves.

The main characteristic of a super-repository is that it records the history of multiple projects in parallel or how we call them, a *software project ecosystem.* Conforming to this definition, both the extremes of a developer's project portfolio and the collection of projects in sourceforge can be considered super-repositories. However, the analysis that can be done on each of them and the questions that need to be answered are different. In our work until now, we have studied super-repositories which belong to a single organization and for which the histories of the rpojects are stored in a SToRE versioning system (2).

3. The Small Project Observatory

The vehicle that drives our super-repository research is the online visualization platform: *The Small Project Observatory*[1] (SPO). It implements a catalog of visualization perspectives that are relevant in the context of understanding super-repositories. SPO (see Figure 1) is interactive, presents multiple live perspectives on the system, provides details on demand and comes with a rich set of filters to address complexity. It obtained the first place in the Innovation Awards competition at ESUG 2007[2].

3.1. *Who is interested in exploring super-repositories?*

In our work we consider super-repositories as first class entities. We believe that there are benefits from analyzing them and understanding them and understanding them as

[1] Available online at http://www.inf.unisi.ch/phd/lungu/spo
[2] 15th International Smalltalk Conference - http://www.esug.org

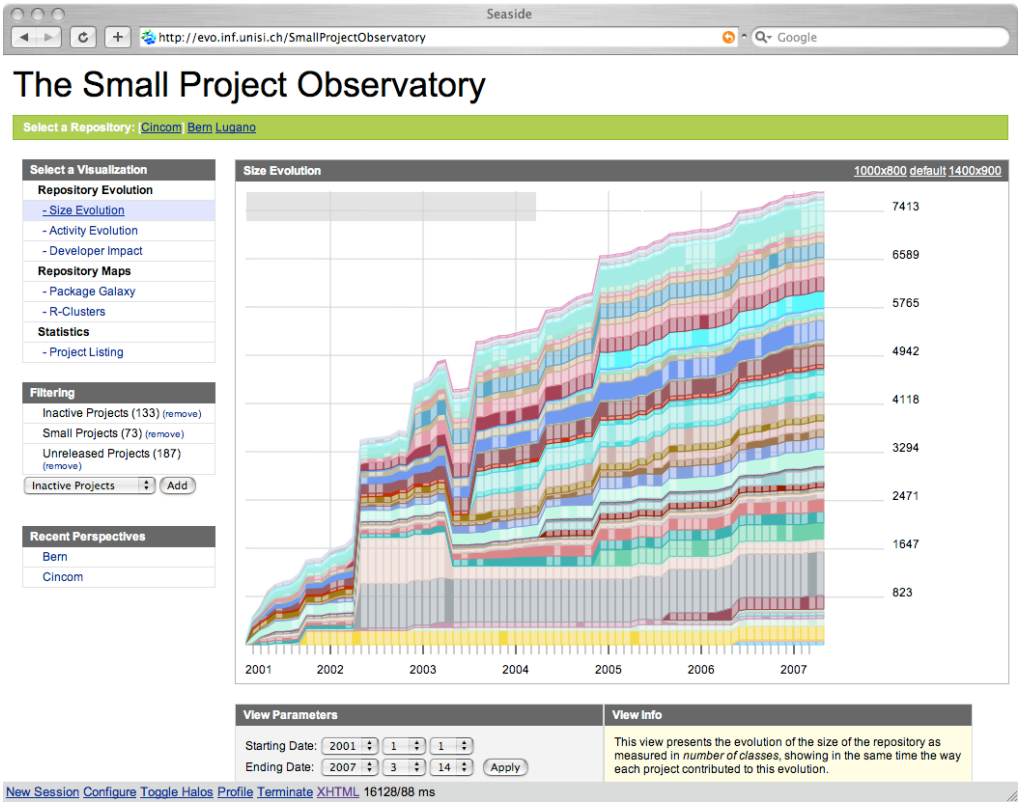

Fig. 1. The user interface of the Small Project Observatory

a whole. Three main categories of stakeholders that should be interested in information about the evolution of the code in a super-repository are project managers, developers, and quality assessment.

Project Managers may ask questions such as "how do teams work?", "how do projects evolve?", or "who has worked on a similar project already?". Organizational charts only show the team structure in a static, and often poorly maintained, form. Revealing the activity and collaboration of developers and the projects they work on, shows how the actual work is being performed (11) and how the collaborations between developers evolved over time. Moreover, since in general successful projects need to continuously change (17; 18), a project manager needs to be up to date with how projects change and what their current status is.

Developers may have questions such as "who should I ask if I want to learn about this feature?", "what dependencies does the system I am working on have and to which applications?", or "what do applications on which my application depends on look like, and what is their current status?". One important source of information for developers, especially for newcomers to a project, are other developers. Thus, developers need to know whom to ask (3). Also, not only the details of a particular project are relevant, but also the inter-project dependencies are important. For example, in the case of a framework, it is important to know who the clients are so that they can be updated.

Similarly, when an application is built out of components, developers need to know what components have changed. In the open-source context there are also developers looking for interesting projects they can contribute to. Since not all of them have equal chances of success, it is useful to gain insights about the evolution, activity and the people involved regarding a particular project.

Quality assessment is interested in the continuous supervision of the evolution of the projects in the super-repository. In this context, alerts can be defined based on various heuristics. One example of heuristic is: "*if a developer who recently joined the company is adding a dependency between two code modules which is contrary to the status-quo, the dependency should be tagged as needing review, and it should be added automatically to the next code review.* Another type of analysis which can benefit from the continuous monitoring of the system is searching for code patterns that appear multiple times inside a project or across projects. Having detected such code patterns can be a first step towards reusing them. Metrics are an important tool in quality assesment (16). However, one problem with metrics is that there are no universal rules for defining thresholds for all the software systems. By mining the information inside an ecosystem the organization can tune the tresholds for its own metric-based alerts.

3.2. *Interactivity in SPO*

The interactive view. The central view displays a specific perspective on a super-repository. In Figure 1 we see the growth in size over a period of 5 years of the projects in one of our case studies; each colored layer in the view represents a different application. The view is interactive in the sense that the user can select and filter the depicted projects, obtain contextual menus for the graphical elements in the view, navigate between various perspectives, or change the time interval of interest. The graphical view is not an image but an interactive SVG element in which every component can be interacted with.

Multiple Perspectives. SPO provides multiple perspectives on a repository such that a user can choose the ones which are appropriate for the type of analysis he needs. The *available perspectives* panel presents the list of perspectives, some of which we will discuss in the article.

Filters. Given the sheer amount of information residing in a super-repository, filters need to be applied on the super-repository data. The panel lists the active filters (in this case only multi-authors projects are depicted in the interactive view), and the user can choose and combine other filters. A user can also apply filters through the interactive view, for example by removing a project or focusing on a specific project using the contextual menu.

3.3. *The architecture of SPO*

Figure 2 presents the architecture of SPO. The main elements are briefly described:

Import and Automatic Update. The import module is responsible for interfacing with the super-repository and pulling data from it. Currently the only type of super-repository that we support is the one for Store. One of the reasons is that we knew that for Store there were no visualization mechanisms. The second reason is that store keeps track of information about the inter-project dependencies so we obtained that

information for free. Currently, importing the data about a few hundred projects can be done in a matter of minutes.

The Internal Representation. At a given moment, in the system there can be multiple super-repositories loaded. Each of them is represented in memory by a model which keeps track of the multiple versions of projects and packages as well as developers and their activity.

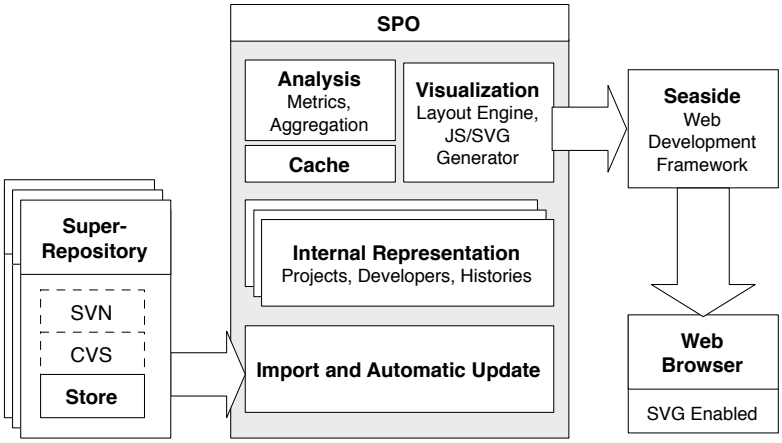

Fig. 2. The architecture of SPO

Alalysis. The analysis module is responsible with computing statistics, metrics and any other derived information from the repository.

Cache. Due to the highly interactive and exploratory nature of the tool, SPO generates dynamically all the web pages and all the visualizations they contain. In order to speed up the view generation the cache module caches all the information that is needed in order to speed-up the view generation process. Even so, the first time a certain type of analysis is done, the user will observe the performance problem.

Visualization. The visualization module takes as input information from the internal representation, analysis and cache modules and generates views from it. The module contains the layout engine and the SVG generator. The Javascript interaction code is generated dynamically for every view. Some of the views, use externally the *dot* graph layout engine [3].

Seaside. Seaside is a web application framework which suppports interaction by emphasizing a component based approach to web applicatoin development. Seaside offers a unique way to have multiple control flows on a page, one for each component (6).

4. Visualizing Super-Repositories

We have applied SPO on multiple project repositories - some public open-source, some research groups and some industrial ones. In Table 1 we provide a brief numerical overview of these repositories.

[3] http://www.graphviz.org/

The oldest and largest of them is the Open Smalltalk Repository hosted by Cincom Systems. The next two are maintained at the Universities of Bern and Lugano, in Switzerland. The last one is a repository maintained by Soops BV, a Dutch software development company. The data provided in Table 1 needs to be considered with care as the numbers are the result of a simple project counting. Super-repositories accumulate junk over time, as certain projects fail, some die, short-time experiments are performed, etc. This is inherent to the nature of super-repositories, and only supports the claim that they need to be understood in more depth.

Super-Repository	**Projects**	**Classes**	**Contributors**	**Active Since**
Cincom	288	19.830	147	2000
Bern	190	10.600	76	2002
Lugano	43	2.088	11	2005
Soops	249	11.413	20	2002

Table 1
The analyzed super-repositories

In the following we present a few of the visual perspectives provided by SPO.

4.1. *Inter-project Dependency*

This perspective presents the static dependencies between projects in a super-repository. Such an overview pinpoints the critical projects in a company.

There are multiple ways of computing the dependencies between projects. In the case-studies that we have been working on up to now, the dependencies are specified directly in the versioning system (22).

The layout of the projects is hierarchical in such a way that the projects which are mostly depended upon are at the bottom. Various metrics computed for the individual projects can be mapped on the color of the nodes representing the project.

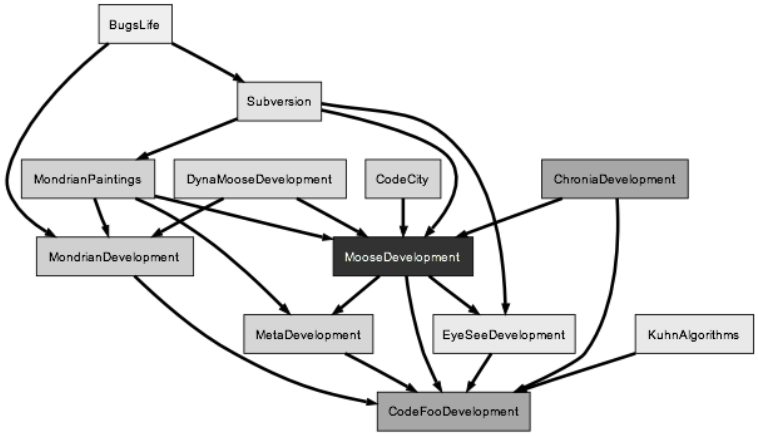

Fig. 3. Inter-project dependencies between a subset of projects in Bern

Figure 3 shows the dependencies between the projects which were active during the month of June 2007 in the Bern super-repository. The convention for the color is that the darker the shading of the project the older it is. The view shows that the oldest project from the projects which are still active is also the one on which the most projects depend on. The project in this case is MooseDevelopment, the reengineering flagship of the Bernese research group.

4.2. *Developer Collaboration*

This perspective shows how developers collaborate with each other across project boundaries withing a super-repository

We say that two developers collaborate on a certain project if they both make modification to the project for a certain number of times above a given threshold. Based on this definition, we construct a *collaboration graph* where the nodes are developers and the edges between them represent projects on which they collaborated.

To represent the collaboration graph for a super-repository we draw the collaboration graph using a *force-based layout algorithm* which positions connected nodes together (10). Thus, developers which collaborate will be positioned closer together. The color of the nodes representing developers is obtained by mapping various metrics selected from the user interface. The color of an arc between two nodes is fixed and represents the project on which the two developers collaborate. If two developers collaborate on more than one project, there will be multiple arcs between them, each one with its corresponding color.

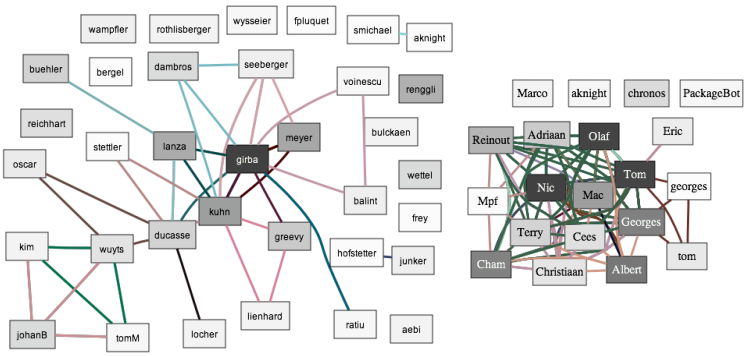

Fig. 4. Developer Collaboration in Bern and at Soops

Figure 4 presents the collaboration perspectives in both the Bern and the Soops super-repositories. The intensity of a node is proportional with the overall activity in the repository of the corresponding developer.

One can see that the two organizations have very different ways of developing software. Although the collaboration perspective in Bern shows a coupled community with many people collaborating on different projects, the density of collaboration in Soops is impressive - to the point where our layouting algorithm was almost overwhelmed. Indeed, after talking to the people at Soops we found out that the policy in the company is that after a certain amount of time, people switch projects such that everybody gets to work on every project.

On the other hand, in the case of Bern, where there were no rules imposed on the collaboration between the developers, *collaboration patterns* are more visible. For example the two extremes of collaboration are what we called *loners* - the people who work alone on their projects - and *mavericks* - developers who collaborate with multiple people on multiple projects. Figure 4 has instances of both types.

4.3. *Developer Activity Lines*

This perspective presents a visual summary of the periods when developers were active in the organization.

Each contributor to the super-repository has an associated activity line which summarizes his activity by marking the periods in time when (s)he was committing changes to the super-repository.

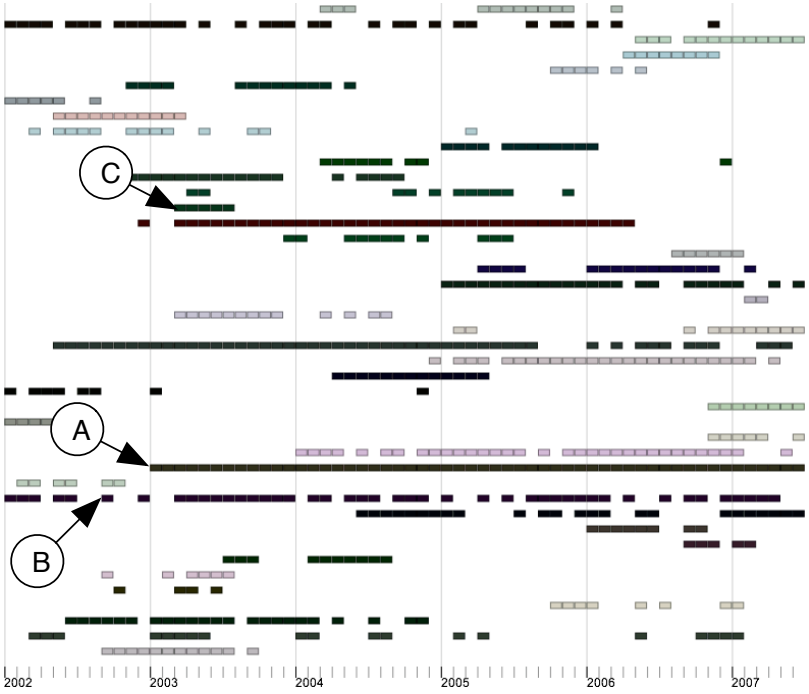

Fig. 5. Developer ActivityLines perspective for the Bern.

Figure 5 presents the history of developer contributions in the Bern super-repository between 2002 and 2007. The figure reveals that the majority of the contributors are active for short periods of time (*e.g.,* C), such as the master students who work on their thesis project. There are also developers who contribute for long periods of time (e.g., A and B), mostly PhD students and Post-docs. In terms of continuity we see that some developers contribute intermittently (B) while others contribute continuously (A and C).

4.4. *Metric Evolution*

This perspective illustrates the evolution of the projects in the super-repository with respect to various metrics.

The visualization principle is that of a stacked graph. The horizontal axis shows time, while the vertical presents a metric that can be chosen from the user interface. The time interval of interest is divided in months, but can be divided also in days or weeks.

Each project has a specific color and is represented as a surface. All the project surfaces are stacked to provide an overview of the total super-repository size evolution. The order in which they are stacked is chronological starting with the oldest projects at the bottom.

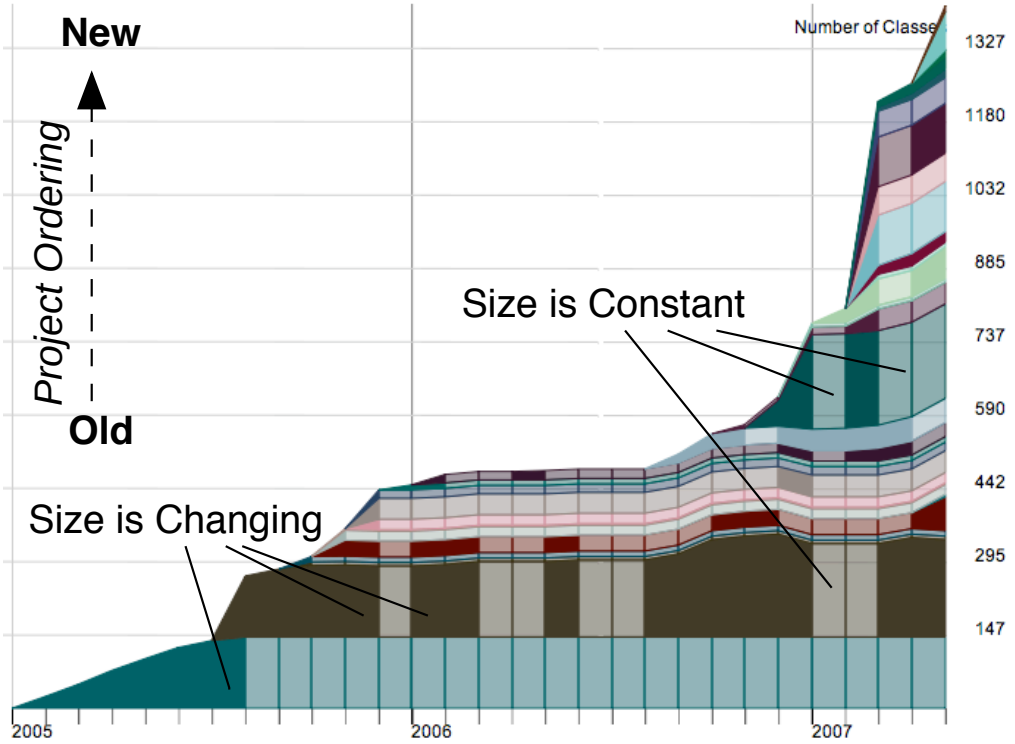

Fig. 6. Size Evolution perspective of the Lugano Super-repository (2005 - 2007)

Figure 6 illustrates the *Size Evolution* perspective on a subset of the projects from the Lugano super-repository between 2005 and 2007. Since we are working with projects written in object-oriented languages, we consider *number of classes* to be a good estimator for the evolution of the size of the projects (13).

The view emphasizes both the evolution in size and the specific time intervals when each project's size changes: the brightness of the project color is higher in the periods when the size remains constant. With this convention we can read from Figure 6 that the oldest project in the repository, has been discontinued after an initial and steady increase as opposed to the next-oldest project which kept growing with a few periods of stagnation.

5. Tool Building Experience

5.1. *Validation*

The first thing we did in order to validate the usefulness of our tool was to use it ourselves. We used it on several open-source super-repositories and we found that it helped discover the developer relationships that existed in those repositories as well as provide insight into the evolution of the projects in the super-repository. Section 4 presents a few examples of information that can be recovered. However, currently there are no guidelines that would suggest a methodological approach to the super-repository exploration. Part of the future work is the write a methodology which would present the ways in which the tool can be used to analyze a super-repository.

However, we also wanted to validate the tool also in an industrial context. While looking for an industrial case-study for our tool we approached Soops b.v, a Dutch software company specialized in Smalltalk, if we could analyze their super-repository using SPO. Due to privacy reasons they denied, but offered instead to install the tool on their own, experiment with it themselves, and report back. We present their experience in detail in (22).

5.2. *Usability*

The experiment with Soops was the first time that we handed over one of our tools away to be tested without our presence. Although we did not have control over the experiment we were satisfied to see that the developers were interested in using the tool and reporting on its usage. However, as soon as they tried to apply it, they discovered that the way they were defining their projects was using a convention that we did not foresee and we had to adapt the tool. The lesson learned is that one needs to be ready to adapt the tools to the peculiarities of the case studies.

A second source of information about usability aspects were informal studies in which we asked colleagues to use the tool and recorded their interactions with the tool. From the analysis of the movies we learned that interaction modes which to us seemed intuitive, proved not to be so for other users. For example, when clicking on the name of a developer in any of the views, the users expected to see a new view with details about the author, but the action that was carried out was to keep the same view and add a filter to remove all the projects that did not belong to that author.

5.3. *Flexibility*

Each time we presented the tool to a colleague, sooner or later, we got the question: *but could you represent that differently?* or *could you visualize that other type of information too?*. This is an instance of a more general type of problem: our users are smart. In Section 3.1 where we present the potential users of our tool it is evident that most of them are technically savvy. For such users, there will be times when they will want to explore a view which is not already implemented, or modify slightly a certain view. In order to accomodate such a scenario, we need to provide more flexibility in view building

and customization. Currently, the only way in which one can implement a new view is by writing a new view class. One possible approach is to let the views be declaratively defined, such as in our work on the Mondrian visualization engine (24).

5.4. *Developing for the web*

One of the reasons for implementing SPO as an online tool was because we wanted to experiment with the possibilities that web applications offer when it comes to visualization and interaction. In our opinion, the benefits availability and ease of update that derive from having an online tool for reverse engineering compensate for the limitations that doing graphics in the browser impose. We present here a few of the issues related to providing interactive visualization with SVG and Javascript.

Scalable Vector Graphics (SVG) is an XML specification and file format for describing two-dimensional vector graphics, both static and animated. SVG offers anti-aliased rendering, pattern and gradient fills, sophisticated filter-effects, clipping to arbitrary paths, text and animations.

SVG is a W3C specification and most of the recent versions of the major browsers support it by default (Firefox 1.5+, Opera 8+, Safari 3) and for the others plugins are available (the most notable is Internet Explorer which supports a specific version called VML). However, although all browsers support it, not all the browsers have the same speed in rendering it and this makes the user experience hard to predict. To illustrate this, we wrote a simple javascript script which calculates the rendering speed of various browsers. We run the script in OS X on a Powerbook G4 running at 1.5GHz with 1G of RAM. The differences betwen the browsers are very large and presented in Table 2:

Browser	**Elements per second**
Safari 311	77
Firefox 2.0.0.4	220
Opera 9.50	595

Table 2
Different browsers render svg at very different speeds

This simple benchmark shows two of the greatest limitations of SVG: the amount of visual elements that one can count on rendering is limited (at least currently) and the user experience is hard to predict as the timings will be different for different users using different system configurations. Moreover, we did encounter problems with the same pop-up menu being rendered different in two different browsers. These limitations are probably part of reasons for which, at the time of writing this article, the trend for interactive visualization applications is biased towards using Adobe's Flash platform.

The interaction part (mouse-over effects, selection, pop-up menus) were implemented usnig Javascript. In terms of expresivity and flexibility, Javascript is a very powerful programming language. The problem that we have encountered when writing Javascript code was the lack of a good integrated development environment and debugging tools. The Firebug plugin for the Mozilla Firefox browser is the best debugging tool we found.

We conclude that using SVG and Javascript both the graphics and interaction are satisfactory and part of our future work will be integrating some of the other software visualization perspectives that we have been working on up to now into SPO.

6. Conclusions and Future Work

In this paper we have presented the Small Project Observatory, an online visualization tool aimed at the visualization and analysis of super-repositories. We presented some of the visualization perspectives that SPO offers at super-repository level and showed that super-repositories contain information about the organization as well as about the code that is produced by the organization.

We believe that super-repository visualization and analysis is a promissing research direction and we plan to continue working on it. Some of the directions in which we would like to focus our efforts are improving the flexibility of the tool as well as providing finer-grained information and supporting other types of super-repositories besides Store.

Analyzing fine-grained information. The visual perspectives that we presented show information only about the elements that are relevant at the abstraction level of an entire super-repository: projects, developers, inter-project relationships. However, there are cases in which the application should support navigating to a lower-level of abstraction such as an inter-module dependency perspective inside a project or the individual method calls that are responsible for an inter-project dependency. Our current super-repository model currently does not support such fine-grained information. We are currently working to extend it and allow the navigation down to the code level.

Supporting other types of super-repositories. We mentioned in the related work section the work of D'Ambros et al.on the online framework for analysis and visualization they call Churrasco. Churrasco supports the online visualization of information recovered from SVN and CVS repositories as well as Bugzilla information. They only analyze single projects but we are considering collaborating with them and integrating features form Churrasco and SPO.

Last but not the least, we want to continue our work by researching new perspectives and implementing new ways of visualizing super-repositories.

References

[1] E. J. Chikofsky, J. H. C. II, Reverse engineering and design recovery: A taxonomy, IEEE Softw. 7 (1) (1990) 13–17.

[2] Cincom, Team Development with VisualWorks. Cincom Technical Whitepaper (2000).

[3] D. Cubranic, G. Murphy, J. Singer, K. Booth, Hipikat: a project memory for software development, Software Engineering, IEEE Transactions on 31 (6) (2005) 446–465.

[4] M. D'Ambros, M. Lanza, A flexible framework to support collaborative software evolution analysis, in: Proceedings of CSMR 2008 (12th European Conference on Software Maintenance and Reengineering), IEEE Computer Society, 2008.

[5] A. M. Davis, 201 principles of software development, McGraw-Hill, Inc., New York, NY, USA, 1995.

[6] S. Ducasse, A. Lienhard, L. Renggli, Seaside: A flexible environment for building dynamic web applications, IEEE Software 24 (5) (2007) 56–63.

[7] S. Eick, J. Steffen, E. S. Jr., Seesoft - a tool for visualizing line oriented software statistics, IEEE Transactions on Software Engineering 18 (11) (1992) 957–968.

[8] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, A. Mockus, Does code decay?

assessing the evidence from change management data, IEEE Trans. Softw. Eng. 27 (1) (2001) 1–12.
[9] P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Müller, J. Mylopoulos, S. G. Perelgut, M. Stanley, K. Wong, The software bookshelf, IBM Syst. J. 36 (4) (1997) 564–593.
[10] T. M. J. Fruchterman, E. M. Reingold, Graph drawing by force-directed placement, Softw. Pract. Exper.
[11] T. Girba, A. Kuhn, M. Seeberger, S. Ducasse, How developers drive software evolution, in: IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution, IEEE Computer Society, Washington, DC, USA, 2005.
[12] T. Gîrba, M. Lanza, S. Ducasse, Characterizing the evolution of class hierarchies, in: Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05), IEEE Computer Society, Los Alamitos CA, 2005.
[13] T. Grba, S. Ducasse, M. Lanza, Yesterday's weather: Guiding early reverse engineering efforts by summarizing the evolution of changes (2004).
URL `citeseer.ist.psu.edu/article/girba04yesterdays.html`
[14] R. Holt, J. Y. Pak, Gase: visualizing software evolution-in-the-large, in: WCRE '96: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96), IEEE Computer Society, Washington, DC, USA, 1996.
[15] M. Lanza, S. Ducasse, Polymetric views - a lightweight visual approach to reverse engineering, Software Engineering, IEEE Transactions on 29 (9) (Sept. 2003) 782–795.
[16] M. Lanza, R. Marinescu, Object-Oriented Metrics in Practice, Springer-Verlag, 2006.
[17] M. Lehman, Programs, life cycles, and laws of software evolution, Proceedings of the IEEE 68 (9) (Sept. 1980) 1060–1076.
[18] M. Lehman, D. Perry, J. Ramil, W. Turski, P. Wernick, Metrics and Laws of Software Evolution - The Nineties View, in: METRICS '97: Proceedings of the 4th International Symposium on Software Metrics, IEEE Computer Society, Washington, DC, USA, 1997.
[19] Y. T. Lin, F. Zou, H. Kienle, H. Müller, A customizable svg graph visualization engine, SVGOpen 2007.
[20] M. Lungu, T. Gîrba, A small observatory for super-repositories, in: Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2007), 2007.
[21] M. Lungu, A. Kuhn, T. Gîrba, M. Lanza, Interactive exploration of semantic clusters, in: 3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005), 2005.
[22] M. Lungu, M. Lanza, T. Girba, R. Heeck, Reverse engineering super-repositories, in: WCRE '07: Proceedings of the 14th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2007.
[23] S. Mancoridis, T. Souder, Y.-F. Chen, E. Gansner, J. Korn, Reportal: a web-based portal site for reverse engineering, Reverse Engineering, 2001. Proceedings. Eighth Working Conference on (2001) 221–230.
[24] M. Meyer, T. Gîrba, M. Lungu, Mondrian: an agile information visualization framework, in: SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization, ACM, New York, NY, USA, 2006.
[25] H. Muller, K. Klashinsky, Rigi: a system for programming-in-the-large, Software Engineering, 1988., Proceedings of the 10th International Conference on (1988) 80–

86.

[26] C. Nentwich, W. Emmerich, A. Finkelstein, A. Zisman, Box: Browsing objects in xml, Software Practice and Experience 30 (15) (2000) 1661–1676.

[27] D. L. Parnas, Software aging, in: ICSE '94: Proceedings of the 16th international conference on Software engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[28] A. J. Perlis, Special feature: Epigrams on programming, SIGPLAN Not. 17 (9) (1982) 7–13.

[29] I. Sommerville, Software engineering (5th ed.), Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[30] M.-A. D. Storey, H. A. Müller, Manipulating and documenting software structures using SHriMP Views, in: Proceedings of ICSM '95 (International Conference on Software Maintenance), IEEE Computer Society Press, 1995.