Reverse Engineering Super-Repositories

In Proceedings of Working Conference on Reverse Engineering (WCRE 2007)

Mircea Lungu, Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland

Tudor Gîrba
Software Composition Group
University of Bern, Switzerland

Reinout Heeck
Soops BV
The Netherlands

Abstract

Reverse engineering and software evolution research has been focused mostly on analyzing single software systems. However, rarely a project exists in isolation; instead, projects exist in parallel within a larger context given by a company, a research group or the open-source community. Technically, such a context manifests itself in the form of super-repositories*, containers of several projects developed in parallel. Well-known examples of such super-repositories include SourceForge and CodeHaus.*

We present an easily accessible platform which supports the analysis of such super-repositories. The platform can be valuable for reverse engineering both the projects and the structure of the organization as reflected in the interactions and collaborations between developers. Throughout the paper we present various types of analysis applied to three open-source and one industrial Smalltalk super-repositories, containing hundreds of projects developed by dozens of people.

1. Introduction

Reverse engineering has been defined by Chikofsky and Cross [3] as "the process of analyzing a subject [software] system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or at a higher level of abstraction".

Indeed, most reverse engineering research is concerned with answering a number of questions on software systems which are closely related to these goals. A great variety of analysis techniques have been created (*e.g.,* metrics[2, 17], visualization[1, 8], clustering[14], architecture recovery[20, 21]) and implemented either in stand-alone tools, or as part of integrated environments.

In the recent years, two interconnected factors have given a new drive to the research field, namely (1) the open source phenomenon, because it led to an increased availability of software systems to be analyzed, and (2) the research topic of "mining software repositories" which deals with techniques to exploit the information contained in versioning systems for evolution analysis.

In this paper we argue that despite the recent advances which made these field as a whole flourish, two issues are being largely ignored:

1. Many reverse engineering techniques are implemented in stand-alone tools. The tools, ranging from simple sets of scripts to full-fledged reengineering environments, such as Moose and Bauhaus, are applied on the systems that need to be analyzed, the results are retrieved, and reasoned on. Accessibility and usability are often poorly addressed concerns in this context, i.e., installing and applying such tools in a productive way requires technical expertise and is often only performed by the tool developers themselves. This often leads to the scenario, where companies, potentially interested by specific software analysis tools and techniques, give up on applying them because of the tools' poor usability and accessibility.
2. Software systems are seldom developed in isolation. On the contrary, many companies, research institutions and the open-source scene deal with software repositories existing in parallel, hosted on dedicated servers[1]. We are faced with *super-repositories*, that is repositories of repositories. In an industrial context such super-repositories represent the *assets* of a company, and besides the evolution of the software systems themselves, a super-repository also contains information about which developers worked on which projects at which time, to what extent and collaborating with whom. Indeed, this added information makes it important to the company to understand what its super-repository contains and how it evolves.

In this article we present a platform which offers a unique and easily accessible entry point to super-repositories in order to facilitate their comprehension. The platform, dubbed Small Project Observatory[2] (SPO), is an interactive web portal accessible through a standard web browser. It offers various means to analyze, visualize and interact with the data contained in a

1 SourceForge for example currently hosts more than 100,000 projects.

2 In the given context, the adjective *small* may be considered a bad pun: its origin lies in the used implementation language (Smalltalk).

super-repository. We claim that it is useful in a variety of contexts: when an open-source contributor is searching for interesting projects to contribute to, when a project manager wishes to supervise multiple projects, or when a new employee wants to understand the "treasure trove" of software that the company has been developing over the years.

We distinguish between two types of super-repositories, (1) repositories that are dedicated to a particular language such as RubyForge[23], CodeHaus[4] and StORE[26], and (2) repositories that are language agnostic such as SourceForge[25] and GoogleCode[12]. Although most of the discourse can be generalized to any of these repository types, in this article we focus our attention on the first category and look at three open-source and one industrial super-repositories which contain each the history of several dozens to hundreds of applications written in Smalltalk.

In Table 1 we provide a brief numerical overview of these repositories. The oldest and largest of them is the Open Smalltalk Repository hosted by Cincom[3]. The next two are maintained at the Universities of Bern and Lugano, in Switzerland. The last one is a repository maintained by the company Soops BV, located in the Netherlands. The data provided in Table 1 needs to be considered with care as the numbers are the result of a simple project counting in the repositories; however super-repositories accumulate junk over time, as certain projects fail, die off, short-time experiments are performed, etc. This is inherent to the nature of super-repositories, and actually only adds to the insight that super-repositories need to be understood in more depth.

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

Who should analyze super-repositories? We argue that different stakeholders are interested in analyzing super-repositories for different tasks. Here we identity three categories of users that benefit from a platform such as SPO, namely *project managers*, *developers*, and *researchers*. Each of these has different reasons to analyze super repositories with respect to specific questions:

3 http://smalltalk.cincom.com

1. **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 [18], a project manager needs to be up to date with how projects change and what their current status is.

2. **Developers** may have questions such as "who should I ask if I want to do that?", "what dependencies does the system I am working on have and to which applications?", or "what do applications on which my application depends 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 [7]. 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 on the evolution, activity and the people involved regarding a particular project.

3. **Researchers** want to identify case studies and extract high level lessons. An easily accessible platform which helps in identifying the appropriate case studies, is a valuable asset and helps not only saving time in the face of the myriads of available systems, but also fosters the research field as approaches can be cross-validated on the same case-studies.

In the remainder of the article we show how our Small Project Observatory (SPO) can help in answering many of these questions by using it in the context of an industrial and three open-source super-repositories.

Structure of the paper. In Section 2 we briefly present the functionalities of SPO and then in Section 3 introduce a catalog of super-repository visualization perspectives that SPO offers. We then present an experience report of using SPO on an industrial super-repository in Section 4. In Section 5 we discuss our approach. We then outline related work in Section 6 and conclude the paper in Section 7.

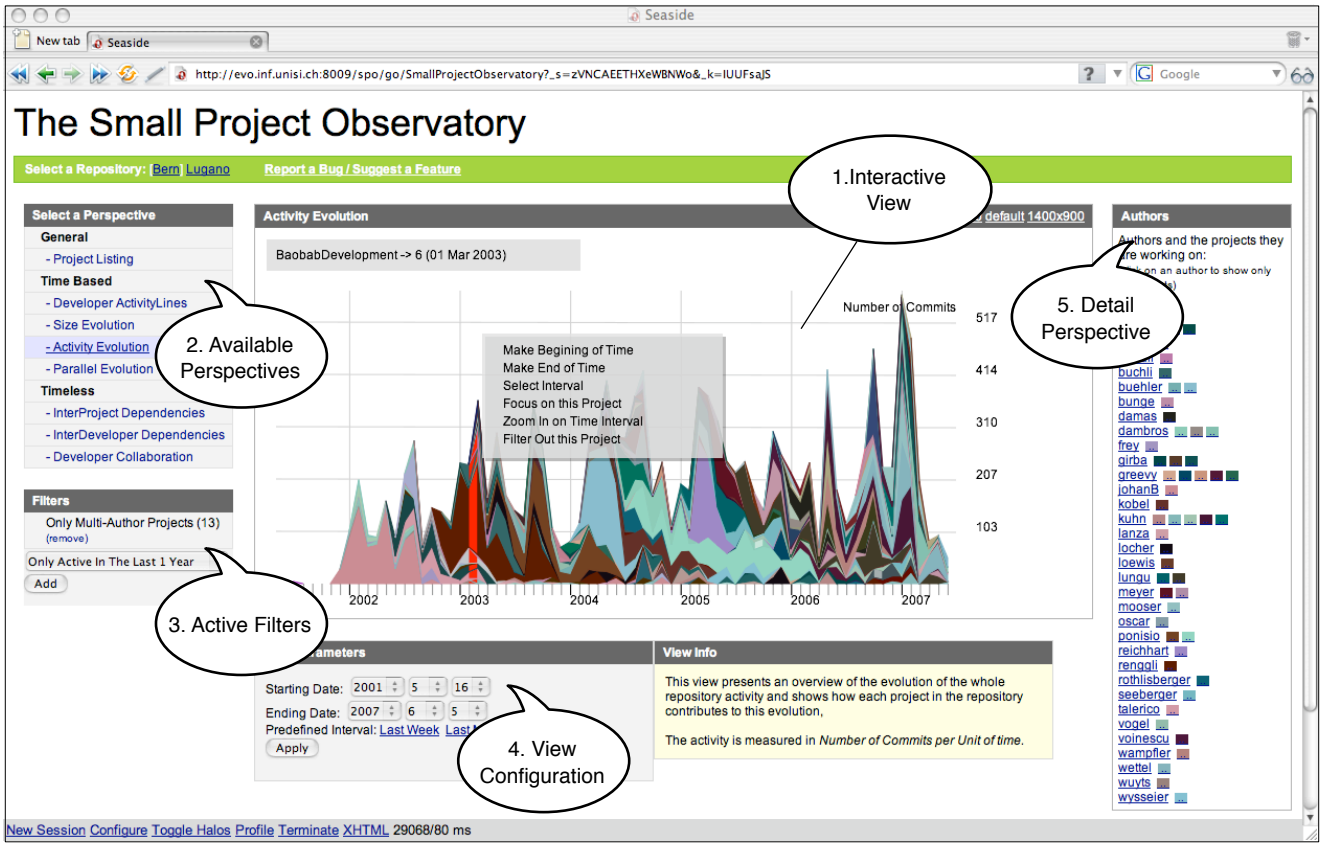

Figure 1. The Interface of The Small Project Observatory

2. The Small Project Observatory

Figure 1 presents The Small Project Observatory[4] within the Opera web browser being used on the Bern super-repository. SPO is a highly interactive web application, and here we present a few of the interaction modes.

The interactive view. The central view displays a specific perspective on a super-repository. In Figure 1 we see the activity (measured in terms of commits to the repository) over a period of 5 years. 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 projects or navigate between various perspectives. Figure 1 presents the contextual menu obtained when the user selects a given project. The view can be configured in terms of the displayed time interval through a selection mechanism available in the view configuration panel (marked as 4).

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 (marked as 2) 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 marked as (3) 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.

Detail perspectives. Providing details on demand is a way of coping with complexity[24]. To the right of the exploration view there are detail panels (marked as 5) which provide additional information on the view or on the selected elements in the view. In Figure 1 the detail panel presents the list of developers which are involved in the projects in the view and the projects they are involved in.

4 A demo version of The Small Project Observatory is available at www.inf.unisi.ch/phd/lungu/spo/

3. Super-repository Perspectives

The Small Project Observatory is implemented as a service which maintains an up-to-date model of a super-repository. Based on this model a multitude of analyses can pe performed. This section presents the types of analyses by presenting the perspectives offered by The Small Project Observatory, and describe how they can be interpreted.

Size Evolution. This perspective illustrates the evolution of the projects in the super-repository with respect to various metrics. The visualization principle, used with success by Wattenberg in other applications [29] is to assign to each project a specific color, and represent it as a surface where the horizontal axis shows time and the height of the surface is given at every point by a certain metric computed at the respective time in the life of the project. Since we are working with projects written in object-oriented languages, we consider Number of Classes to be a good estimation [10] for the evolution of the size of the projects.

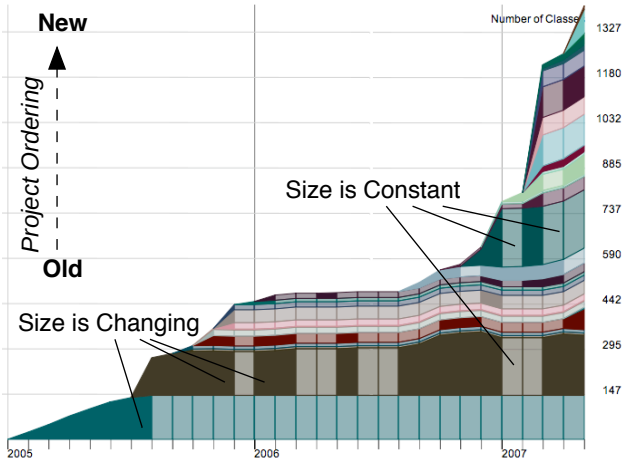

Figure 2. Size Evolution perspective of the Lugano Super-repository (2005 - 2007)

Figure 2 illustrates the concept of the size evolution perspective on a subset of the projects from the Lugano super-repository between 2005 and 2007. The time interval of interest is divided in months, but can be divided also in days or weeks. 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. The view not only emphasizes the evolution in size but also emphasizes 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 infer from Figure 2 that the project at the bottom, the oldest in the repository, has been discontinued after an initial and steady size increase.

Activity Evolution. The Activity Evolution perspective complements the previous perspective by depicting the activity within the super-repository over time, i.e., it renders the effort spent by developers. To measure activity we use the number of commits.

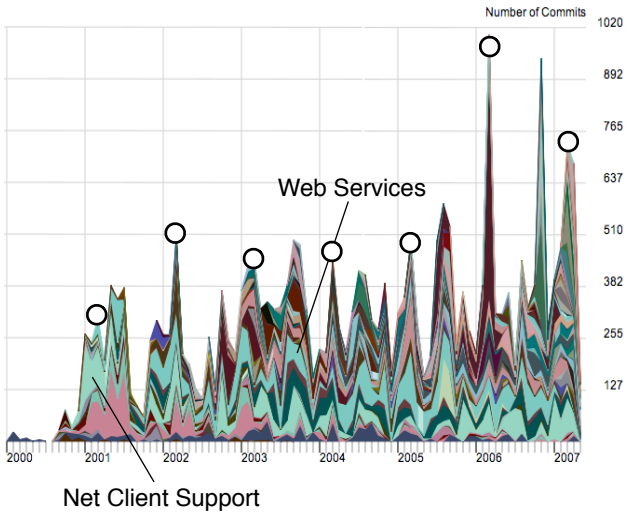

Figure 3. Activity Evolution perspective of the Cincom Super-repository (2000 - 2007)

Figure 3 presents the evolution in time of the aggregated activity in the Cincom super-repository between 2000 and 2007. The units on the horizontal axis are months. A first observation related to Figure 3 is that there are several projects which are continuously active for long periods of time. The two marked are Net Client Support and Web Services, two of the oldest projects in the repository. Another observation regarding activity is that the alternance of peaks and valleys presents some repetitive patterns with drops in August and December. This is easily attributable to the holidays seasons. Another interesting phenomenon is the increase in productivity at the beginning of the year, marked by circles. Although we have observed the same phenomenon in the Bern super-repository we have no theory on the underlying cause.

Parallel Evolution. This perspective combines the two previously presented ones into one single perspective, and is mostly useful during drill-down phases.

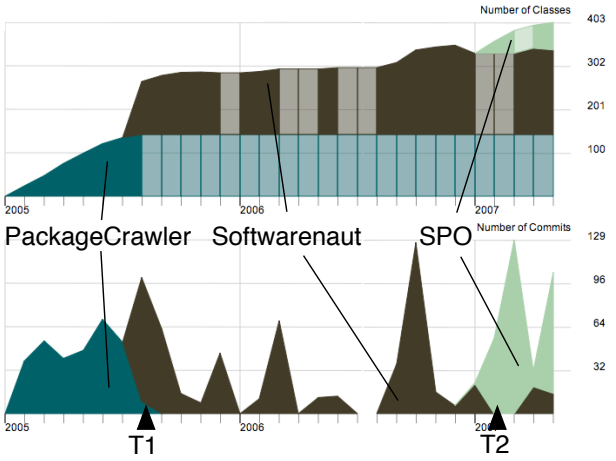

Figure 4. Parallel Evolution Perspective of 3 projects (2005 - 2007).

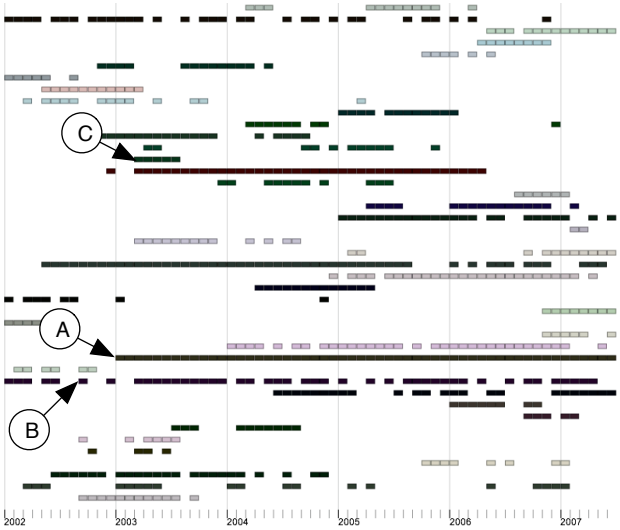

Figure 5. Developer ActivityLines perspective of the Bern super-repository (2002-2007).

Figure 4 was obtained by filtering in only the projects in the Lugano super-repository for which one of the authors (*i.e.,* Lungu) was the main developer. We see three projects (*i.e.,* PackageCrawler, Softwarenaut, and SPO) corresponding to various research directions explored during the PhD of one of the authors. The view shows that at mid-2005 (T1) the activity on PackageCrawler stops completely and the activity on Softwarenaut begins. What is not visible in the figure is the fact that Softwarenaut took several components from PackageCrawler and continued from there. The second observation is that at the beginning of 2007 (T2) the focus of the development effort changes from Softwarenaut to SPO although the work on Softwarenaut continues. [5]

Developer Activity Lines. The Developer Activity Lines perspective presents a visual summary of the developer activity in the repository. Each contributor to the super-repository has an associated activity line which sumarizes his activity by marking the periods in time when (s)he was comitting changes to the super-repository.

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 several developers who contribute for long periods of time (such as the ones marked A and B in the figure), 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).

Inter-project Dependency. The Inter-project dependency perspective presents the static dependencies between projects of a super-repository. Such an overview pinpoints the critical projects in a company, or projects that *cannot* die. 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 project representations.

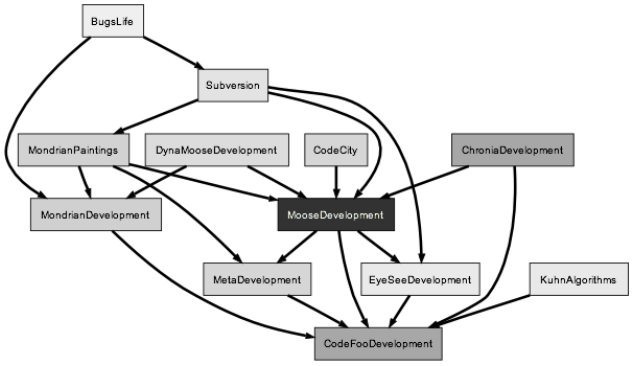

Figure 6. Inter-Project Dependencies between the projects active in the last month in Bern

5 The activity spike at the end consists in several changes needed to support the current paper.

Figure 6 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 SCG research group.

Developer Collaboration. This perspective shows how developers collaborate with each other within a super-repository, i.e., across project boundaries. 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. We call this metric the *developer commit count (DCC)*. Based on this information 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 graph using a *force-based layout algorithm* which clusters connected nodes together and offers an aesthetically pleasing layout [9]. Thus, developers which collaborate will be positioned closer together. The intensity of a node's color can be proportional to other metrics. Because an arc between two nodes represents the project on which the two nodes collaborate, the arc has the color of the respective project.

Figure 7 presents the collaboration perspective of the Bern super-repository. We considered only developers with a DCC count > 15. The intensity of a node is proportional with the overall activity in the repository of the node (*i.e.,* the darker the node, the more active is the corresponding developer). The perspective allows for a classification of developers based on their type of collaboration.

We observed three types of developers, *loners, collaborators,* and *hubs*. Loners work alone on projects. Figure 7 shows that in the analyzed repository this type of user is very well represented, probably given to the "lonely" nature of the development performed during a PhD or Master's. Collaborators work with others on few projects. As an example, developer "lienhard" (point A) from Figure 7 is involved in a single project in which other two developers work. Hubs collaborate on many projects. For example, developer "wuyts" (point B) from Figure 7 has connections to multiple developers and is involved in several projects.

Overall, the Bern super-repository shows a large and tightly coupled community. Indeed the Berne research group has worked on many facets of reverse engineering during the past years, leading to a myriad of tightly coupled tools, capped by the Moose reengineering environment. This might be a result of Conway's law which states that organizations that produce systems are constrained to produce designs which are copies of those organizations [6].

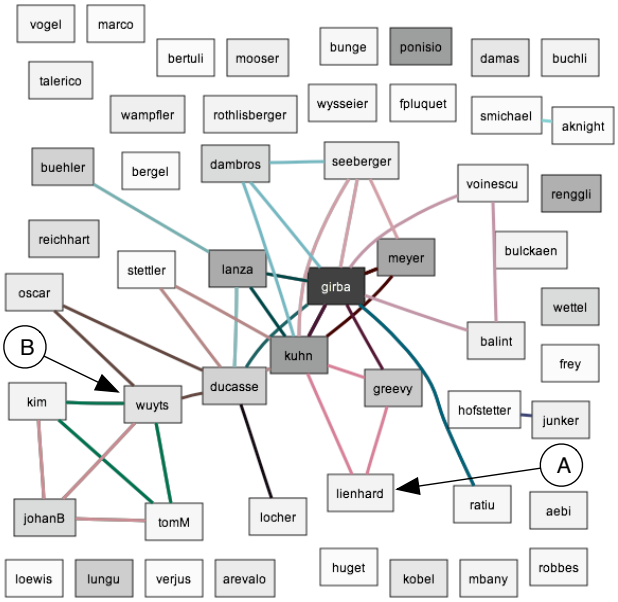

Figure 7. Developer Collaboration perspective of the Bern super-repository

4. An Experience Report at Soops BV

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 the interpretations:

The development team at Soops has been using Store since it was first released in the 5i version of VisualWorks. Over time we found that bundles[6]*, were too cumbersome to be used in an agile process, particularly in an* everybody owns the code *setting, so Soops has since declined to use bundles to group code packages, instead we opted to use a different mechanism called* lineups *[19]. In our case the repository contains both lineups and bundles, where bundles are created by parties outside Soops and lineups relate to code created at Soops. The first thing that needed to be done was to adapt SPO to support lineup analysis. An initial analysis run reports 249 projects in the repository, adjusting the filters to only show activity in the past year re-*

6 Bundles are the Store mechanism for projects. The term will be used interchangeably with projects in this section

duces this number to 188. All further analyses are restricted to the past year.

Developer Activity Lines. *The first thing that we wanted to see was the history of developer activity. Looking at Figure 8 some things stand out.*

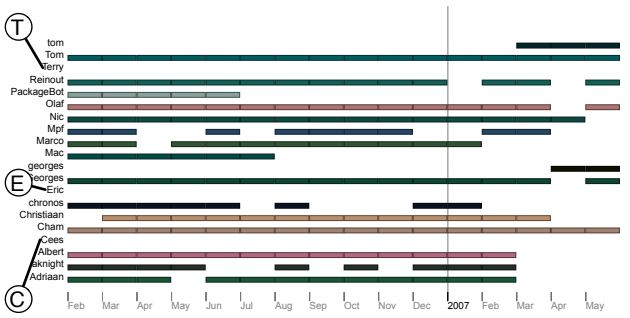

Figure 8. Developer Activity Lines during the last year in the Soops repository

User 'Mpf' is only occasionally contributing to the repository. The reason is that he is outsourced to customers of Soops and hence shows gaps in his commit behavior. Packagebot only committed early in the year, this reveals a breach of Soops' publishing protocol: the PackageBot login was not intended to be used for committing, but this was not enforced by access controls. Three of the developers (marked E, C and T) show no activity over this period of time. These three developers were external hires in earlier years, their names still appear in the graph because the projects they worked on are still under active development.

Developer Collaboration. *To learn more about the developer structure we switch to the Collaborations perspective. Figure 9 shows a couple of disconnected developers, of those 'aknight' and 'chronos' refer to authors of third-party packages. PackageBot should have never committed as explained earlier. Marco is a developer who writes test suites, he does not contribute application code so he rarely commits into the same packages as the developers. 'Mpf' is in the same position as Marco but has helped develop the test tool itself as well, which shows as some of his collaboration edges in the graph. Eric was maintaining a single project, mainly together with Tom. The remaining people show strong collaboration which reflects the situation at Soops where developers regularly switch between projects. Trying to untangle this central knot of collaborations by switching to a hierarchical layout gives little extra clarity, collaboration appears to be abundant.*

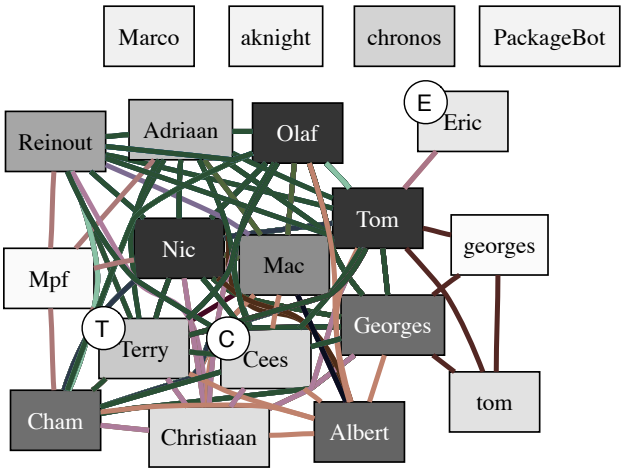

Figure 9. The Inter-Developer Collaboration perspective shows abundant collaboration

Activity Evolution. *As we have seen in the previous view, several of the developers are not part of the core team of the company so we filtered their projects. On the remaining projects we generated an Activity Evolution perspective, shown in Figure 10.*

Looking at the commit activity there is one project standing out as being 'large', mousing over it reveals that this is the 'Jun' project, a third party OpenGL access layer that has been used at Soops for research purposes. Jun is not distributed in a format compatible with the Store repository. Scripts are available on the web to convert Jun to Store but this proved to be cumbersome, quite a large number of commits were required before a properly loading project bundle was created. Since Jun is not core to Soops' products, we elide it from the graph using the filters supplied by SPO (displayed in part (b) of the figure).

The graph now shows a more regular spread of activity over the projects, interpreting the graph requires 'mousing over' the various parts to see which project names they are associated with. This reveals that bundles are drawn as the bottom layers of the graph and lineups as the top layers. Since at Soops this dichotomy aligns closely with the third-party vs Soops's software we can concentrate on these two halves separately. Looking at the bottom half we see three surges of activity (marked as A) on July 2006, March 2007 and May 2007. Mousing over reveals that the brown swaths are related to the 'Base VisualWorks' bundle, these activity surges show at what times Soops published a VisualWorks release into this repository. The first two peaks correspond to builds internal to Cincom[7] that Soops has access to, the

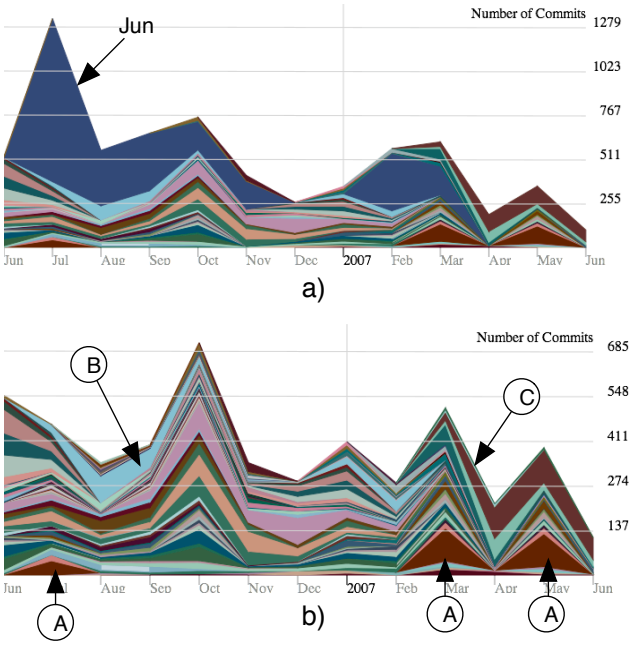

Figure 10. Activity Evolution in the Soops Repository between June 2006 and June 2007 with (a) and without Jun (b).

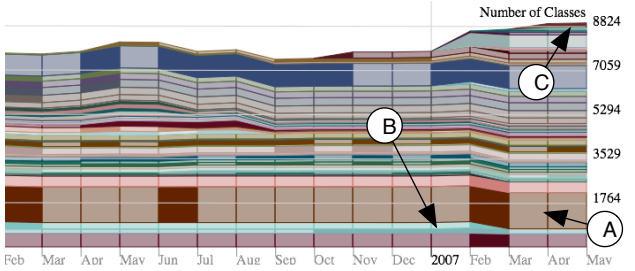

Figure 11. Size Evolution in the Soops repository

last one signifies the official release of VisualWorks 7.5. Further inspection of the bundle names reveals that these commits in 2006 only comprise two bundles ('Base VisualWorks' and 'Tools-IDE') present in the base Smalltalk image, whereas the two activity peaks in 2007 comprise many more bundles related to externally loadable libraries delivered with the VisualWorks product.

Moving our attention to Soops specific projects in the top of the graph we see two that stand out by their activity: the light blue swath with its activity peak in August 2006 (marked as B) and the brown ribbon spanning from February to June (marked as C). Mousing over the interactive diagram reveals that the first one is related to a 'plugin' created by Soops to communicate with a third-party product. This project had many technological challenges at lower layers (multi-threaded COM connect) requiring several rewrites of it's core components and this is why the development spanned half a year. Moving on to the brown area at the right this shows to be a major application that has only recently been ported from VisualWorks version 3 to version 7.5. Since version 3 uses another SCM tool (Envy) than 7.5 it has never been committed to this repository until porting the project got underway in February 2007. As can be seen activity on that modernization project has steadily grown since it was ported.

Size Evolution. *Looking at the sizes of projects (Figure 11, again with 'Jun' elided) we can see that the size of the code in the repository has a general tendency to increase even if there are periods in the lifetime of the super-repository where the size decreases. Looking at the projects in the repository we can see multiple projects which are being touched intermittently, a sign of ongoing maintenance.*

One of the most prominent projects in the figure is the somewhat 'fat' one at the bottom signifying the Cincom product which hardly varies in volume (marker A), except once in march 2007 where it collapses slightly. The light-blue line that disappears in March 2007 (marker B) is the 'Refactoring Browser' tool that has been renamed and assimilated into existing bundles. Oddly SPO shows an overall reduction of code here while we would expect no change of size, merely a different distribution between projects. In the range June - September 2007 we see that Soops' code also decreases in size, this can in part be attributed to changes in code generating tools that were introduced, sparser code was generated for the 'Soops-API' project. The reasons for other declines of size are not readily apparent, trolling through the release comments shows that code for one project 'Market Configuration Server' has been moved to other packages. It seems that SPO no longer counts this code as part of a project, this could be due to the fact that Lineups don't carry enough information to automatically discern between code contained in a project and code that is a mere prerequisite. The bands on top of the graphic starting in February (marker C) relate to the project mentioned earlier that was ported from VisualWorks 3.

7 The supplier of VisualWorks Smalltalk.

5. Discussion

The Experiment. 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. We received usability feedback which we plan to incorporate in future versions. The first lesson learned is that we have to be ready to adapt our tools to make them fit the particularities of the case studies. As mentioned in the previous section, we had to adapt our tool to the way that the Soops developers define projects.

Another lesson that we have learnt is that different people need different views. While The Small Project Observatory has been only tested on open-source systems, when applied in the Soops context not all views proved to be useful. For example, one of the Inter-Project Dependency view was not useful due to too much noise generated by too many dependencies between the projects.

Interpretation Pitfalls. It is tempting to derive conclusions after seing a perspective. It might seem that a developer with a high commit count is more useful to the company. However, people have different ways of working and a developer committing many small changes might still be less instrumental to the company than one who commits less frequently but works on an important project in the system. This is why the perspectives should not be considered alone but in a larger context.

Developer Collaborations. The way the collaboration relationship is defined can be improved. For example, we could evaluate the quality –not only the quantity– of changes the developers make. Another problem related with the developer collaboration relatinship is that although it is a dynamic property of a super-repository currently the Developer Collaborations perspective represents the state of the relations between the developers at a single point in time, i.e., in the last version of the system. It would be interesting to visualize the evolution of these relationships.

Privacy. Some of the data that we visualize involves delicate issues such as developer activity. In the case of open-source systems this information is available but in an industrial context this informatin has to be treated with attention. We are grateful to Soops for providing us with information about their development environment.

6. Related Work

Several approaches rely on visualization to understand the history of software systems, but most of them focus on one system only. Lanza and Ducasse devised the Evolution Matrix to focus on how classes change [16]. Rysselberghe *et al.* used a simple plot diagram to identify change patterns [27]. Wu *et al.* made use of the spectrograph metaphor to reveal hot periods in a project [31]. Girba *et al.* devised the Ownership Map to show how developers changed the system [11]. Voinea *et al.* propose multiple visual perspectives on the entire project history [28]. Rötsche and Krickhaar [22] presented a system for supervising the evolution of the refactoring process of a large scale industrial system.

There are only few projects which analyze entire repositories. One such project is the FlossMole project which provides for download a database compilation of open-source projects from Sourceforge and several other repositories [5]. Weiss performed a very interesting analysis of all the projects in SourceForge, however his visualizations are statistical in nature[30]. Kawaguchi *et al.* used semantic analysis to categorize software systems in open-source software repositories [13]. They provide a tool that categorizes the projects and labels the categories. Kuhn *et al.* also used a similar approach to analyze relationships between projects [15]. As opposed to our work, these approaches have been applied on one version only.

German proposed the analysis of software distributions as a means to understand the relative importance of software packages [**?**]. Distinct from supre-repositories, software distributions only contain stable, released software packages. Based on the characteristics of the dependency graph German proposes metrics that quantify the success of various packages.

7. Conclusions

In this paper we argue for the importance of super-repository visualization and present The Small Project Observatory, a platform that supports super-repository analysis. Our contributions can be summarized as follows:

- We presented a set of super-repository visualization perspectives and exemplified them on three open-source super-repositories,
- We implemented the visualizations in a tool called The Small Project Observatory that we have briefly presented, and
- We presented an experience report of using The Small Project Observatory in an industrial setting.

Acknowledgments. We would like to thank Daniel Ratiu, Romain Robbes and Jochen Wuttke for feedback on previous drafts of this article. We are grateful to Soops BV for trying out and reporting on the usage of SPO. We also acknowledge the support of the Swiss National Science Foundation for the project "NOREX — Network of Reengineering Expertise" (SNF Project IB7320-110997).

References

[1] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, 29(4):33–43, 1996.

[2] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[3] E. Chikofsky and J. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, Jan. 1990.

[4] Open-Source Project Repository With A Strong Emphasis on Java. http://codehaus.org. http://codehaus.org/.

[5] M. Conklin, J. Howison, and K. Crowston. Collaboration using ossmole: a repository of floss data and analyses. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.

[6] M. E. Conway. How do committees invent? *Datamation*, 14(4):28–31, Apr. 1968.

[7] D. Cubranic and G. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings 25th International Conference on Software Engineering (ICSE 2003)*, pages 408–418, New York NY, 2003. ACM Press.

[8] S. Demeyer, S. Ducasse, and M. Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In F. Balmas, M. Blaha, and S. Rugaber, editors, *Proceedings of 6th Working Conference on Reverse Engineering (WCRE '99)*. IEEE Computer Society, Oct. 1999.

[9] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 1991.

[10] T. Gîrba, S. Ducasse, and M. Lanza. Yesterday's Weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In *Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 40–49, Los Alamitos CA, Sept. 2004. IEEE Computer Society.

[11] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2005)*, pages 113–122. IEEE Computer Society Press, 2005.

[12] Open-Source Project Hosting by Google. http://code.google.com/hosting.

[13] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue. Mudablue: An automatic categorization system for open source repositories. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*, pages 184–193, 2004.

[14] R. Koschke and T. Eisenbarth. A framework for experimental evaluation of clustering techniques. In *Proceedings of the International Workshop on Program Comprehension, IWPC'2000*. IEEE, June 2000.

[15] A. Kuhn, S. Ducasse, and T. Gîrba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, Mar. 2007.

[16] M. Lanza and S. Ducasse. Beyond language independent object-oriented metrics: Model independent metrics. In F. B. e Abreu, M. Piattini, G. Poels, and H. A. Sahraoui, editors, *Proceedings of the 6th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, pages 77–84, 2002.

[17] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

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

[19] Travis Grigs' Blog: Line Ups as Reported by Reinout Heeck. http://www.cincomsmalltalk.com/userblogs/travis/blogView?showComments=true&entry=3265388740.

[20] M. Pinzger. *ArchView – Analyzing Evolutionary Aspects of Complex Software Systems*. PhD thesis, Vienna University of Technology, 2005.

[21] C. Riva. *View-based Software Architecture Reconstruction*. PhD thesis, Technical University of Vienna, 2004.

[22] T. Rötschke and R. Krikhaar. Architecture Analysis Tools to Support Evolution of Large Industrial Systems. In *Proc. IEEE International Conference on Software Maintenance (ICSM 2002)*, pages 182–193, 10 2002.

[23] RubyForge the home of open source Ruby projects. https://rubyforge.org. http://rubyforge.net/.

[24] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Visual Languages*, pages 336–343, College Park, Maryland 20742, U.S.A., 1996.

[25] A Development and Download Repository of Open Source Code and Applications. http://www.sourceforge.net/.

[26] Team Development with VisualWorks. Cincom Techincal White Paper. Cincom Technical Whitepaper.

[27] F. Van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings 20th IEEE International Conference on Software Maintenance (ICSM '04)*, pages 328–337, Los Alamitos CA, Sept. 2004. IEEE Computer Society Press.

[28] L. Voinea, J. Lukkien, and A. Telea. Visual assessment of software evolution. *Science of Computer Programming*, 365(3):222–248, 2007.

[29] M. Wattenberg. Baby names visualization, and social data analysis. In *Proceedings of 2005 IEEE Symposium on Information Visualization (INFOVIS 2005)*, pages 1–6, 2005.

[30] D. A. Weiss. A large crawl and quantitative analysis of open source projects hosted on sourceforge. In *Research Report ra-001/05, Institute of Computing Science, Pozna University of Technology, Poland, 2005. At http://www.cs.put.poznan.pl/dweiss/xml/publications/index.xml*, 2005.

[31] J. Wu, R. Holt, and A. Hassan. Exploring software evolution using spectrographs. In *Proceedings of 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 80–89, Los Alamitos CA, Nov. 2004. IEEE Computer Society Press.