Promises and Perils of Porting Software Visualization Tools to the Web

Marco D'Ambros, Michele Lanza, Mircea Lungu, Romain Robbes
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

Abstract

Software systems are hard to understand due to the complexity and the sheer size of the data to be analyzed. Software visualization tools are a great help as they can sum up large quantities of data in dense, meaningful pictures. Traditionally such tools come in the form of desktop applications. Modern web frameworks are about to change this status quo, as building software visualization tools as web applications can help in making them available to a larger audience in a collaborative setting. Such a migration comes with a number of promises, perils and technical implications that have to be taken into account before starting any migration process.

In this paper we share our experiences in porting two such tools to the web and discuss the promises and perils that go hand in hand with such an endeavour.

1. Introduction

Developing tools is an important part of software engineering research as they provide a proof-of-concept for an approach. Further, the tool itself can be considered a research contribution. However, tools remain often at the stage of prototypes, not maintained anymore after the corresponding article is published. Little effort is spent in making tools long-lived and used in an industrial context, with a number of notable exceptions such as the Moose reverse engineering framework [7], visualization tools such as Rigi [19], and recommender systems like Mylyn [13].

In the vast majority of cases, tools do not survive after a research has been published and concluded. In his keynote address at ICSE 2009[1], Carlo Ghezzi stated that a survey of all the papers that appeared in TOSEM[2] between 2000 and 2008 showed that 60% of them dealt directly or indirectly with tools. However, of those, only 20% were actually installable, let alone functional.

In the past years, we have developed a number of software visualizations tools, such as CodeCrawler [14], Softwarenaut [15], BugCrawler [4], Evolution Radar [5], Bug's Life [6], Churrasco [3], and The Small Project Observatory [16]. Many of these tools are available, but some effort from the accidental user to make them work is required. This certainly decreases their adoption and impact. The solution to this problem is to exploit the web and some of the modern technologies that are available. We see the web as an opportunity to improve the accessibility and adoption of research prototypes: If a tool is available as a web application then there is no installation and the cost for people to "give it a try" is minimal.

However, developing web-based software visualization tools is not easy, and comes with a number of promises to embrace and perils to avoid. In this paper we discuss our experience in building two web-based software visualization tools and distill a number of considerations that need to be made if one wants to port such tools to the web. The goal of this paper is to provide guidance to researchers who want to move their (visualization) tools to the web, or want to create new web-based tools from scratch.

Structure of the paper. In Section 2 we briefly describe the web-based software visualization tools that we have developed, Churrasco and Spo. Based on our experience in building these tools, we distill a number of promises and perils for porting such tools to the web in Section 3, which we discuss in Section 4. We then look at related work on web-based analysis and visualization tools in Section 5 and conclude in Section 6.

2. Churrasco and SPO

In the last years we have developed two web-based software visualization tools: *Churrasco* and *SPO* (short for the *Small Project Observatory*), available respectively at http://churrasco.inf.unisi.ch and http://spo.inf.unisi.ch.

2.1. Churrasco

Churrasco [3] is a web-based platform for collaborative software analysis with the following characteristics:

- It provides an easily accessible interface over a web browser to create models of software systems and of

[1] The 31st ACM International Conference on Software Engineering
[2] ACM Transactions on Software Engineering and Methodology

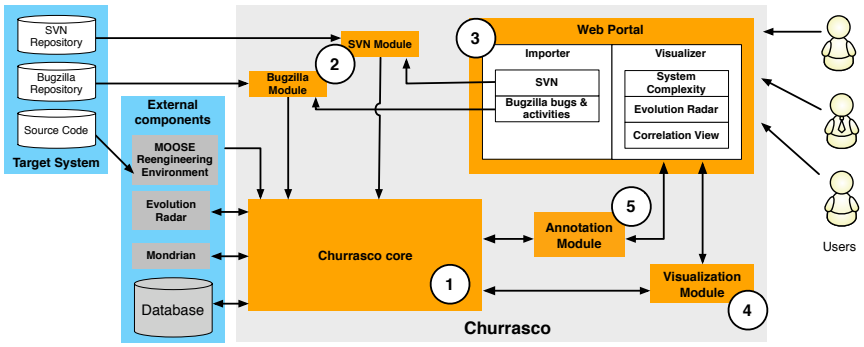


Figure 1. The architecture of Churrasco.

their evolution, and to store them in a database for subsequent analysis.

- It provides a set of visual analyses and supports collaboration by allowing several users to annotate the shared analyzed data.
- It stores the findings into a central database to create an incrementally enriched body of knowledge about a system, which can be exploited by subsequent users.

Figure 1 depicts Churrasco's architecture, consisting of:

1. *The core* connects the various modules of Churrasco and external components. It includes the internal representation of a software system's evolution and manages the connection with the database to write models imported from the web interface and to read models to be visualized in the web portal.
2. *The Bugzilla and SVN modules* retrieve and process the data from SVN and Bugzilla repositories.
3. *The Web portal* represents the front-end of the framework (developed using the Seaside framework [8]) accessible through a web browser. It allows users to create the models and to analyze them by means of different web-based visualizations.
4. *The Visualization module* supports software evolution analysis by creating and exporting interactive Scalable Vector Graphics (SVG) visualizations. The visualizations are created by two external tools: Mondrian [18] and the Evolution Radar [5] . The visualization module converts these visualization to SVG graphics. To make them interactive within the web portal, Churrasco attaches AJAX callbacks to the figures, allowing server-side code to be executed when the user selects a figure.
5. *The Annotation module* supports collaborative analysis by enriching any entity in the system with annotations. It communicates with the web visualizations to depict the annotations within the visualizations.

Figure 2 shows an example of a System Complexity visualization [14] rendered in the Churrasco web portal. The main panel is the view where all the figures are rendered as SVG graphics. The figures are interactive: Clicking on one of them will highlight the figure (red boundary), generate a context menu and show the figure details (the name, type and metrics values) in the figure information panel on the left.

A key idea behind Churrasco is collaboration: Each model entity can be enriched with annotations to (1) store findings and results incrementally into the model and to (2) let different users collaborate in the analysis of a system in parallel or succession. Annotations can be attached to *any* visualized model entity, and each entity can have several annotations. An annotation is composed of the author who wrote it, the creation timestamp and the text. Since the annotations are stored in a central database, any new annotation is immediately visible to all the people using Churrasco, thus allowing different users to collaborate in the analysis. This is further supported by other features in Churrasco, such as the "Recent annotations" panel, which displays the most recent annotations added, together with the name of the annotated entity. By clicking on it the user can highlight the corresponding figures in the current visualization. The "Participants" panel lists all the people who annotated the visualizations, i.e., people collaborating in the analysis. When one of these names is clicked, all the figures annotated by the corresponding person are highlighted in the view, to see which part of the system that person has been working on.

2.2. The Small Project Observatory

The Small Project Observatory (SPO from hereafter) is an interactive web application to visualize and analyze software ecosystems. A software ecosystem is a collection of

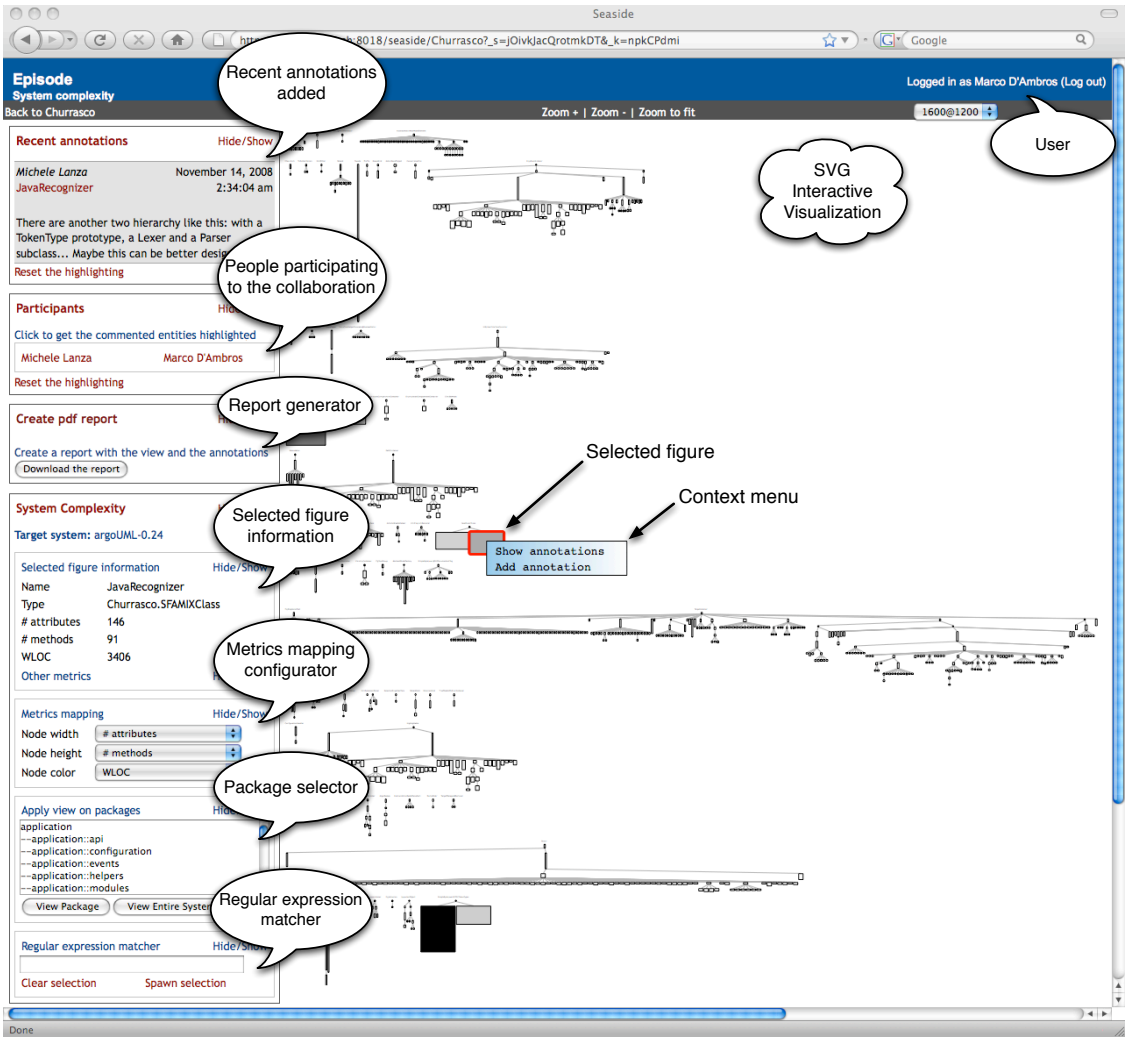


Figure 2. A screenshot of the Churrasco web portal showing a System Complexity visualization of ArgoUML (http://argouml.tigris.org).

software projects which are developed and evolved together in the same environment. Information about a software ecosystem can be found in a super-repository, a 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.

Figure 3 shows a screen capture of SPO during one of our ecosystem case studies. SPO provides multiple views on a repository. The user chooses the ones which are appropriate for the type of analysis he needs. The Views panel (labeled 1) presents the list of all the available views.

The central view (labeled 2) displays a specific perspective on a super-repository. In this case it is a table that presents metrics about the projects in the super-repository. The view is interactive: The user can select and filter the available projects, sort the displayed projects, obtain contextual menus for the projects or navigate between various perspectives.

Given the sheer amount of information residing in a super-repository, filters need to be applied to the super-repository data. The top-right panel lists the active filters (labeled 3). In the case of Figure 3 the only active filter is "In-house projects". The user can choose and combine existing 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.

Figure 4 shows another perspective over an ecosystem: A visualization applied to the software ecosystem that is

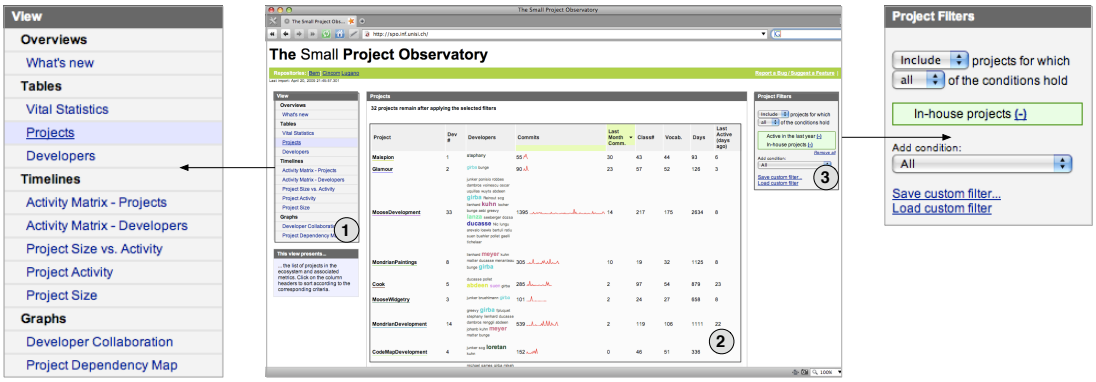


Figure 3. The user interface of SPO.

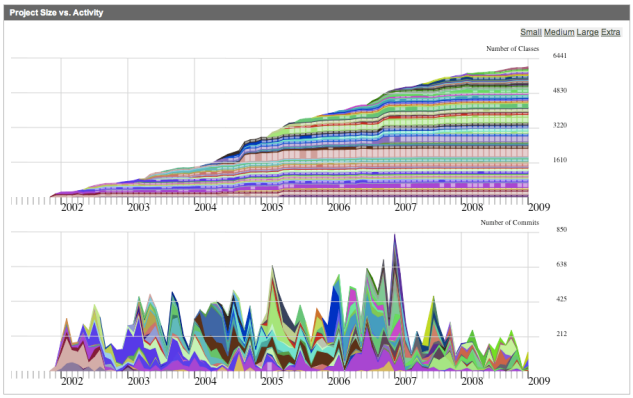


Figure 4. Size and activity evolution for the projects in the SCG ecosystem.

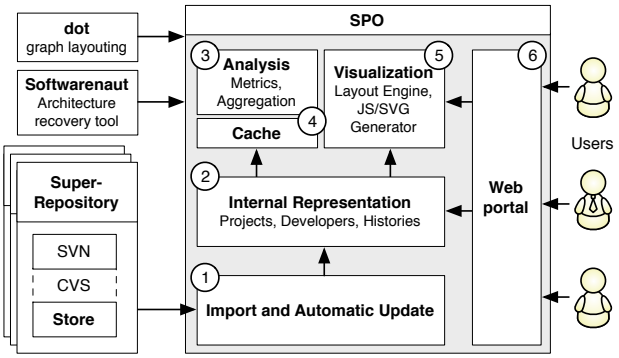


Figure 5. The architecture of SPO.

hosted by the Software Composition Group from the University of Bern, Switzerland. It presents two timelines displayed in parallel: the growth of the size (top graph) and the fluctuation of the activity (bottom graph). The size is measured in number of classes while the activity is measured in number of commits. The figure shows that size is monotonically increasing while the activity fluctuates over time with regularities and with a general trend being visible. One of the regularities is the dip in activity towards the end of every year and in the summer. This rhythm corresponds to the holiday periods of students. The general trend shows increase in activity until the peak of January 2007 when there are 700 commits. After that date, the overall activity level seems to have fallen.

Figure 5 presents a conceptual diagram of the architecture of SPO. The main components are:

1. *The import module* is responsible for interfacing with the super-repository and pulling data from it. Currently SPO supports two types of super-repositories: one based on SVN and another one based on Store, a specific Smalltalk repository.

2. *The internal representation* contains a repository of ecosystem models.

3. *The analysis module* is computing metrics and any other information that can be derived by analyzing the information in the ecosystem model.

4. *The cache module*. Due to the highly interactive and exploratory nature of the tool, SPO generates dynamically all the web pages and all the visualizations they contain. This module caches all the information that is needed in order to speed-up the view generation.

5. *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, which delegates the layouting to the Dot external tool[3], and the SVG generator.

[3] See http://hoagland.org/Dot.html

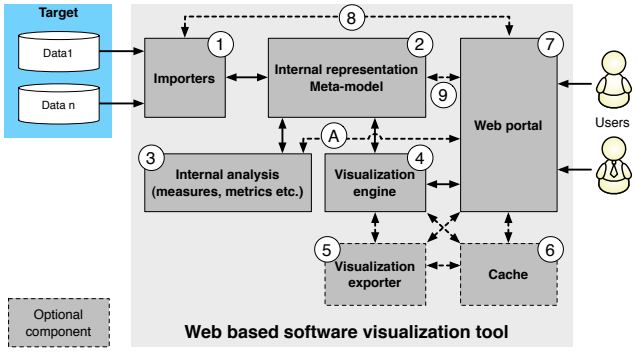


Figure 6. General architecture of a web-based software visualization tool.

6. *The web portal* is the user interface of SPO, built on top of the Seaside web application framework.

2.3. Beyond Churrasco and SPO

Figure 1 and Figure 5 show the architecture of Churrasco and SPO. Based on our experience, we abstracted a general architecture for web-based software visualization tools displayed in Figure 6. Dashed elements are optional components. Software visualization tools provide views on one or more aspects (e.g., source code, bug report, mail archive, etc.) of a software. Therefore, they have an importer module (1) which retrieves the data and stores it according to an internal representation (2). The data is then optionally processed to compute metrics (3) about the considered aspects. The data is finally visualized by means of a visualization engine (4): In case the engine does not produce a web suitable visualization, an exporter (5) is used to create the web visualization. To improve the performances one can use a cache component (6) which avoids recomputing the visualizations. The software visualization tool has a web portal which displays the visualizations (7), imports the data (8), accesses the models (9), and computes the metrics (A).

3. Promises and Perils

In this section, we recall our experience building Churrasco and SPO and extract various aspects of it in the form of promises and perils, summarized in Table 1.

Promise 1 - Availability: Porting software visualization tools to the web makes them more available than desktop applications.

Availability and Privacy
Promise 1. Porting software visualization tools (SVT) to the web makes them more available than desktop applications
Peril 1. Sensible information about software systems should not be available for not authorized people
Collaboration and Performance
Promise 2. Porting SVT to the web eases making them collaborative
Peril 2. Web-based software visualization applications (WBSVA) have to serve large amounts of data to several users, which can be a performance bottleneck impacting all users
Promise 3. WBSVA ease the creation of an incrementally enriched knowledge about a software
Error Handling
Peril 3. WBSVA are single points of failure
Peril 4. Debugging and testing web applications is hard
Promise 4. WBSVA provide feedback about errors
Promise 5. WBSVA make it possible to gather usage data
Development
Peril 5. WBSVA have to tackle cross browser issues
Peril 6. Developing interactive web applications is hard
Peril 7. Web technologies are changing fast
Promise 6. WBSVA can use external tools to perform a number of tasks, exposing only the results as services

Table 1. Summary of promises and perils.

Many research prototypes have problems with respect to their availability. Often such prototypes are hard to install because of compatibility issues, missing libraries, missing documentation etc. Among the various reasons behind the availability problem, one is that researchers do not have the manpower required to create and update documentation, maintain the software, keep the web site (when existing) up-to-date, etc. Moreover, academic research is mostly publication-driven, and not tool-driven, i.e., there is little direct benefit that comes with maintaining tools.

Tracking the evolution of systems and components requires further effort, as compatibility issues occur over time when new versions of components the tool depends on are released. Having the application running on a Web server means that the environment can be frozen, so that supporting the latest version of a component is not a priority.

Indeed, porting research prototypes to the web increases the availability of such tools and avoids installation problems. In the case of both Churrasco and SPO all that needs to be given to users is the url.

Peril 1 - Privacy: Sensible information about software systems should not be available for unauthorized people.

Having a tool available on the web implies that anybody can access it. Web-based software visualization tools might

have access to sensible information about a software system, which should be accessible only by authorized people. For this reason, such tools should provide an authorization mechanism that is not required for desktop applications.

In Churrasco we tackled this problem by letting only registered users accessing the visualizations, and by giving different users different privileges. SPO does not implement authentication yet. As a result, when we approached an industrial partner for a case study on the ecosystem of the company, the partner declined to import their data in the online version of SPO. They installed a local version of SPO on their intranet and performed the analysis themselves.

Promise 2 - Collaboration: Porting software visualization tools to the web eases the process of making them collaborative.

Sharing the data among users naturally leads to collaboration. Virtually all software is nowadays built in a collaborative fashion. This ranges from the usage of SCM systems supporting distributed development, now widely used in practice [10], awareness tools to prevent upcoming conflicts [21], to fully integrated solutions such as IBM's Jazz [12].

Just as the software development teams are geographically distributed, consultants and analysts are too. Analysis tools supporting collaboration would allow different experts with a distinct range of skills to collaboratively analyze a software system from various locations and/or time zones.

Churrasco supports collaboration using a central database: Different users access the same web portal, and analyze the same models of software systems. Users collaborate by annotating the entities composing the models and by looking at other people's annotations. This simple collaboration facility proved useful in a couple of experiments. Improving it via the addition of richer communication channels, such as chat or tagging, is easy to achieve in a web application.

Desktop applications can also support collaboration, but we argue that this is harder to implement. In this case, the various instances of the application need a communication channel among themselves directly in a peer-to-peer fashion or using a centralized server. This leads to networking issues due to firewalls. We are not aware of software visualization tools which support collaboration, but a number of visualization tools in other domains support it [1, 9].

Peril 2 - Performance and Scalability: Collaborative, visual web applications have to serve large amounts of data to several users at the same time, which can be a performance bottleneck impacting all users.

Web applications typically have to serve several users at the same time, and collaborative applications even more so. Depending on the number of users and the type of application, the performance per user might decrease. This is especially true for software visualization applications, where for large datasets both the computation time and the size of the data to be sent to the user's browser might be large, increasing the user's waiting time and therefore the usability of the application.

Visualization must scale up as it is most useful to deal with large amounts of data [22]. Since the visualizations are rendered on the client side, bandwidth can become and issue. For example, in Churrasco an SVG graphic visualizing the ArgoUML software system (ca. 1800 classes) is larger than 1 MB, while SPO generated SVG images going up to 2MB. SPO however reduces the bandwidth by compressing the data to be sent, effectively trading CPU usage for increased bandwidth. In that case the 2MB file was reduced to 150KB.

The standard way of rendering a web visualization is that every time something changes in the page, the whole page is refreshed. In the context menu example, whenever the user clicks on a figure the page changes because a new figure appears, and therefore the page needs to be refreshed to show the menu. Refreshing the entire web page for every action introduces latencies which make the web application slow when it comes to rendering very large SVG files. One way to avoid this problem is to use semantic zoom and details on demand to keep the rendered image small. Churrasco can focus on a single package of a system, while SPO allows the definition of filters. Another possibility is to minimize the page refreshes by using AJAX updates, which refresh only the changed part of the page, as Churrasco does. However, if the use of AJAX has been simplified over time, it is still not trivial.

Concurrent usage is another issue made potentially worse by collaborative work. With Churrasco and SPO we performed two experiments, with 8 participants each, without any performances issue. However, 8 users is a small number which does not allow for general statements.

This peril can be tackled by having several instances of the web application running on several servers, with a web server responsible of dispatching the requests and balancing the CPU and bandwidth loads. Such a solution is standard fare in web applications. However, for research prototypes such a hardware infrastructure is often not available.

Promise 3 - Incremental results: Web-based software visualization tools ease the creation of an incrementally enriched body of knowledge on software systems.

Despite performance and scalability issues, sharing the data paves the way for new possibilities. Results of analyses on software systems produced by tools are often written

into files and/or manually crafted reports, and have therefore a limited reusability. To maximize their reuse, analysis results should be incrementally and consistently stored back into the analyzed models. This would allow researchers to develop novel analyses that exploit the results of previous analyses, leading to a cross-fertilization of ideas and results. It can also serve as a basis for a benchmark for analyses targeting the same problem (i.e., by tagging entities that a candidate analysis should detect, we can compare approaches), and ultimately would also allow one to combine techniques targeting different problems.

By using a central database where all the models are stored, and by letting users annotate the entities composing the models, the users can store the results of the analysis on the model itself, thus supporting the incremental storage of results. This is supported in Churrasco, and can be easily implemented in other web-based software visualization applications, using a shared central database.

Peril 3 - Single point of failure: Web-based applications are single points of failure.

Excessive centralization reduces the reliability of the application. Web-based applications run on a server, and usually have a unique instance of the application which all the users access. As a consequence, if the application crashes it will lock out all users, i.e., the application represents a single point of failure, whereas in desktop applications each user has a private instance of the application, where a crash does not impact the other users. This peril can be tackled, together with performances, by distributing the computation on several servers for redundancy.

Peril 4 - Debugging and testing: Debugging and testing web applications is hard.

A barrier to develop web applications is the lack of support for debugging. Even if there are some applications like Firebug (http://getfirebug.com) providing HTML inspection, Javascript debugging and DOM exploration, the debugging support is not comparable with the one given in mainstream IDE such as Eclipse. Moreover, the testing of a web-based system is hard to perform, due to the lack of consolidated techniques and supporting tools.

Promise 4 - Feedback: Web-based software visualization tools provide feedback about errors and failures.

If debugging a web application is more difficult than a desktop one, being notified of bugs and deploying the fixes is actually easier. Because of the restricted manpower available when developing them, research prototypes are far from being mature and stable applications. Indeed, researchers do not have the resources to invest a significant amount of time testing their application. These problems impact the usage of the tools and therefore their adoption by other researchers or people from industry. One way to be notified about these issues is to instrument the tool so that if it crashes, it collects information about the run-time scenario and then asks the users to send this information back to the developers. This widely adopted approach requires a significant infrastructure and is therefore mostly used in commercial applications.

By having the tool as a web service, the tool is always running on the server, and therefore the tool developer can be notified of all bugs and failures. Bug fixes also do not need to be distributed to individual users, but are available to all users at once.

Promise 5 - Usage report: Web applications make it possible to gather precise usage data.

Similarly to error notifications, gathering usage data is easy. With desktop applications it is possible to track the number of downloads of a tool, and the tool might be instrumented to send back feedback about how it is used. This is however not straightforward to implement. Web-based applications offer the possibility to exploit standard solutions to the usage statistics problem, such as Google analytics. This allows developers to easily gather usage statistics and infer popular features or usability problems, to continuously improve the tool. As with bug fixes, deploying updates is transparent.

Peril 5 - Browser compatibility: Web applications have to tackle cross browser issues.

Web browsers are a rather diverse crowd, and the fact that a web application works with one browser does not guarantee that it works with other browsers. While many compatibility issues can be solved, such as how CSS (Cascading Style Sheets) are interpreted, others cannot. In these cases the users have to focus on a particular web browser to exploit the full functionality of the web application.

Visualization applications have requirements which make this situation more probable: For instance, Churrasco uses AJAX callbacks to update SVG depictions without refreshing the entire web page. The SVG DOM update in AJAX is supported only by Firefox and, as a consequence, Churrasco is only fully functional with Firefox.

SVG is a W3C specification and most of the recent versions of major web browsers support it: Opera and Safari support it without AJAX update and Internet Explorer supports it through a third party plug-in. However, not all the

browsers have the same speed in rendering it, which makes the user experience unpredictable. To test this, we wrote a simple JavaScript program which calculates the rendering speed of various browsers. We ran the script in OS X on a PowerBook G4 running at 1.5GHz with 1GB of RAM. The differences between the browsers are very large. For example, in one second Opera 9.50 renders 595 polygons while Safari only renders 77. This simple benchmark shows two of the greatest limitations of SVG: The amount of visual elements that one can render is limited (at least currently) and the user experience is hard to predict, as the timings will be different for users with different system configurations. Also, we encountered problems with the same pop-up menu being rendered differently in two browsers.

Other technical choices such as Flash or JavaScript (with APIs such as Processing or the JavaScript InfoVis Toolkit[4]) may alleviate these problems. JavaScript in particular has seen a resurgence of interest among web browser builders who now compete over their JavaScript performance.

Finally, it is not unreasonable to require a widespread browser such as Firefox over Internet Explorer if the benefits of the application are promising enough.

Peril 6 - Interaction: Developing interactive web applications is harder than desktop applications.

Supporting interaction through a web browser is a non-trivial task, and even supposedly simple features, such as context menus, must be implemented from scratch. In Churrasco context menus are implemented as SVG composite figures, with callbacks attached, which are rendered on top of the SVG visualization. In SPO such menus are dynamically generated by Javascript. It is hard to guarantee a responsive user interface, since every web application introduces a latency due to the transport of information.

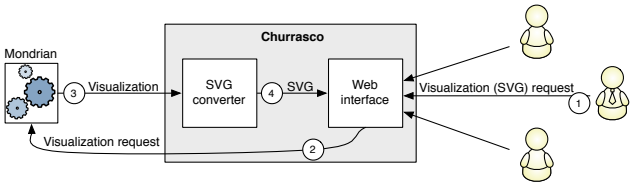
However, libraries of reusable components are quickly developing, such as Scriptaculous and jQuery for Javascript[5], which should alleviate this problem somewhat.

Peril 7 - Rapid evolution: Web technologies are changing fast.

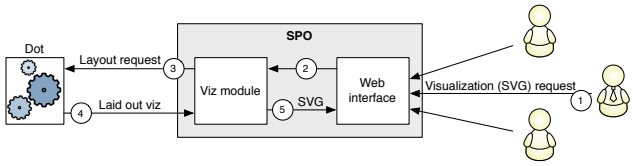
The dust is far from settled in the web technology arena. As we saw above, several technologies (SVG, Flash, Javascript, etc.) are currently competing. These technologies are rapidly evolving: New possibilities are emerging, and the amount of support among browsers varies. This rapid evolution makes it difficult to choose which tools/libraries/technologies to use, and to maintain the web application aligned with the rapidly evolving technologies. Developers must be watchful of new opportunities and potentially capable to switch to newer technologies when needed. We hope that, with time, standard solutions will emerge for highly interactive, graphical web applications.

Promise 6 - Hiding tasks and exposing services: Web-based visualization applications can use external tools to perform tasks, exposing the results as services.

Some aspects of web application development are however easier. Implementing software visualization tools as web applications allows the developer to use external tools in the backend. This is much harder with desktop applications because external tools have to be included in the application distribution, and they should run on the client machine (which might also have installation problems like the application itself). In short, the web application developer has total control over the environment the application is executing in.



(a) Churrasco uses the Mondrian framework in the backend to create visualizations and then it converts them as SVG interactive graphics.



(b) SPO uses the Dot external tool to layout its visualization.

Figure 7. Two examples of using external tools in Churrasco and SPO.

The use of external tools offers a lot of reuse opportunities, such as layout engines. For example, Churrasco reuses two external tools (Mondrian [18] and the Evolution Radar [5]) to create visualizations, which are then converted to SVG by a dedicated module of Churrasco (see Figure 7(a)). This enables us to freely reuse all the visualizations and layouts provided by Mondrian and the Evolution Radar. SPO is dispatching the layouting of its visualizations to Dot, a Unix command line layout algorithm library (see Figure 7(b)).

SPO also exposes the service of Softwarenaut [15], an architecture recovery tool whose visualizations where

[4] Available respectively at: http://processingJS.org and http://thejit.org

[5] Available at respectively http://script.aculo.us and http://jquery.com

adapted to the Web. Moreover, SPO is processing huge amounts of data (entire super repositories) when there are no user connected, i.e., exploiting idle time, caching the results and presenting them on-demand to the users. In this way, SPO is hiding heavy computations and presenting only the results as a lightweight service. Churrasco does the same thing when, given the url of a SVN or Bugzilla repository, it sends an email to the user when the data is imported.

Figure 8 shows how the usage of external tools can be generalized: The web interface gets the request for a visualization and dispatches it to an external tool. The result is then converted in a web-suitable format and sent back through the web interface to the clients' web browsers.

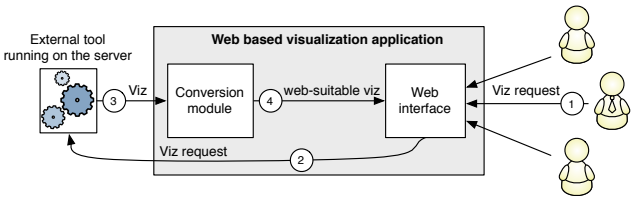


Figure 8. The general schema for using external tools in web-based visualization applications and hide them behind the web interface.

4. Discussion

We argue that in developing a web-based software visualization tool the benefits of the promises are greater than the mostly technical issues and challenges of the perils. In particular, we argue that the most important promises are:

- *Availability*. In the Introduction we observed that 80% of tools presented in TOSEM in the last 8 years are not even installable. The web can improve this situation.
- *Reuse*. We showed that with web applications it is possible to hide tasks and provide services. Porting or creating a web visualization requires a smaller implementation effort, as not only libraries but even entire external tools can be reused.
- *Collaboration*. Collaboration is getting more and more attention both in forward and reverse engineering. We believe that this trend will continue and collaboration will play a key role in these domains in the following years. We discussed how and why, with web applications, supporting collaboration is easier with respect to desktop applications.

To increase their survival chances, every software visualization tool, in the long run, should have a web front-end. This does not require a huge implementation effort because many existing tools can be just reused, and it will increase the accessibility of the application and its adoption.

The perils of developing web applications should be however taken into account. In such a rapidly evolving domain, it is especially important to evaluate which technology fits best the developer needs when it comes to porting or creating a web visualization. Nowadays the choice is among SVG, Flash, Javascript Processing and InfoVis. The choice of the technology should be based on the interaction levels offered and the one needed for the application, how easily it integrates with the existing application / framework (if about porting a visualization), and the debugging facilities available.

5. Related Work

As far as we know, besides Churrasco and SPO there are no web-based software visualization tools. The most related work are software visualization tools which produce outputs readable by web browsers, and web-based software analysis tools without visualizations.

Beyer and Hassan proposed Evolution Storyboards [2], a technique that offers dynamic views. The storyboards, rendered as SVG files, depict the history of a project using a sequence of panels, each representing a particular time period. These visualizations are partially interactive: They only show the names of the entities in the figures. In contrast the views offered in Churrasco and SPO are fully interactive, providing context menus for the figures and navigation capabilities. The Evolution Storyboard is not a web application, but a tool producing SVG files readable by browsers.

Nentwich et al. introduced BOX, a portable, distributed and interoperable approach to browse UML models [20]. BOX translates a UML model in XMI to VML (Vector Markup Language), which can be directly displayed in a web browser. BOX enables software engineers to access and review UML models without the need to purchase licenses of tools that produced the models. As the Evolution Storyboard, BOX is not a web application but a tool which can produce output readable by some web browsers.

Mancoridis et al. presented REportal, a web-based portal site for the reverse engineering of software systems [17]. REportal allows users to upload their code (Java or C++) and then to browse, analyze and query it. These services are implemented by reverse engineering tools developed by the authors over the years. For doing that the authors exploited promise 6 - Hiding tasks and exposing services. REportal supports software analysis through browsing and querying, but does not offer interactive visualizations.

Finnigan et al. developed the Software Bookshelf, a web-based paradigm for the presentation and navigation of information representing large software systems [11].

6. Conclusion

Building software visualization tools for the web is a daunting task that we experienced first-hand when we implemented two web-based tools, Churrasco and SPO. We documented our experiences in the form of promises and perils of such a transition. The transition to the web has a variety of technological consequences making some tasks harder (e.g., debugging, scaling), but some other easier (e.g., error reporting, maintainance). The web is a moving target: Technologies and standards are rapidly changing, and one must regularly assess the technological choices made in the light of changing support accross browsers. If completed, this transition is rewarding: A web-based tool has a greater visibility and potential impact, as people can work with it without needing to install it. A web platform also makes collaboration a more probable possibility, as the costs to implement it are lower than in standalone applications.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "DiCoSA" (SNF Project No. 118063).

References

[1] C. Bajaj and S. Cutchin. Web based collaborative visualization of distributed and parallel simulation. In *Proceedings of the IEEE symposium on Parallel visualization and graphics (PVGS 1999)*, pages 47–54. IEEE Computer Society, 1999.

[2] D. Beyer and A. E. Hassan. Animated visualization of software history using evolution storyboards. In *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 199–210. IEEE CS Press, 2006.

[3] M. D'Ambros and M. Lanza. A flexible framework to support collaborative software evolution analysis. In *Proceedings of CSMR 2008 (12th IEEE European Conference on Software Maintenance and Reengineering)*, pages 3–12. IEEE CS Press, 2008.

[4] M. D'Ambros and M. Lanza. Visual software evolution reconstruction. *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, 21(3):217–232, 2009.

[5] M. D'Ambros, M. Lanza, and M. Lungu. Visualizing co-change information with the evolution radar. *Transactions on Software Engineering (TSE)*, page to appear, 2009.

[6] M. D'Ambros, M. Lanza, and M. Pinzger. "a bug's life" — visualizing a bug database. In *Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis)*, pages 113–120. IEEE CS Press, 2007.

[7] S. Ducasse, T. Gîrba, and O. Nierstrasz. Moose: an agile reengineering environment. In *Proceedings of ESEC/FSE 2005*, pages 99–102, 2005.

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

[9] K. Engel and T. Ertl. Texture-based volume visualization for multiple users on the world wide web. In M. Gervaut, D. Schmalstieg, and A. Hildebrand, editors, *Proceedings of the Eurographics Workshop in Vienna, Austria*, pages 115–124, 1999.

[10] J. Estublier, D. Leblang, A. van der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Transactions on Software Engineering and Methodology*, 14(4):383–430, Oct. 2005.

[11] P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Mueller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. The software bookshelf. *IBM Systems Journal*, 36(4):564–593, Nov. 1997.

[12] R. Frost. Jazz and the eclipse way of collaboration. *IEEE Software*, 24(6):114–117, 2007.

[13] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT 2006/FSE-14)*, pages 1–11, New York, NY, USA, 2006. ACM.

[14] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *Transactions on Software Engineering (TSE)*, 29(9):782–795, Sept. 2003.

[15] M. Lungu and M. Lanza. Softwarenaut: Exploring hierarchical system decompositions. In *Proceedings of CSMR 2006 (10th IEEE European Conference on Software Maintenance and Reengineering)*, pages 349–350. IEEE CS Press, 2006.

[16] M. Lungu, M. Lanza, T. Gîrba, and R. Heeck. Reverse engineering super-repositories. In *Proceedings of WCRE 2007 (14th IEEE Working Conference on Reverse Engineering)*, pages 120–129. IEEE CS Press, 2007.

[17] S. Mancoridis, T. S. Souder, Y.-F. Chen, E. R. Gansner, and J. L. Korn. Reportal: A web-based portal site for reverse engineering. In *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE 2001)*, page 221. IEEE Computer Society, 2001.

[18] M. Meyer, T. Gîrba, and M. Lungu. Mondrian: An agile visualization framework. In *Proceedings of Softvis 2006 (3rd International ACM Symposium on Software Visualization)*, pages 135–144. ACM Press, 2006.

[19] H. A. Müller. *Rigi — A Model for Software System Construction, Integration, and Evaluation based on Module Interface Specifications*. PhD thesis, Rice University, 1986.

[20] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. BOX: Browsing objects in XML. *Software Practice and Experience*, 30(15):1661–1676, 2000.

[21] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: Raising awareness among configuration management workspaces. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 444–454, 2003.

[22] C. Ware. *Information Visualization*. Morgan Kaufmann, 2000.