Telling Stories about GNOME with Complicity

Sylvie Neu, Michele Lanza, Lile Hattori, Marco D'Ambros
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

***Abstract*—Traditionally, the target of software evolution research has been single software systems. However, in the recent years, researchers observed that software systems are often not developed in isolation, but within a larger context: the ecosystem level. Analyzing software evolution at the ecosystem level allows a better understanding of the evolution phenomenon, as the entire development context can be studied. Nonetheless, software ecosystem analysis is challenging because of the sheer amount of data to be processed and understood.**

We present Complicity, a web-based application that supports software ecosystem analysis by means of interactive visualizations. Complicity breaks down the data quantity by offering two abstraction levels: ecosystem and entity. To support a thorough exploration and analysis of ecosystem data, the tool provides a number of fixed viewpoints and the possibility of creating new viewpoints with given software metrics. We illustrate in a case study how Complicity can help to understand the GNOME ecosystem in a bottom-up approach, starting from a single project and contributor towards their impact on the ecosystem.

I. Introduction

Software systems change, and during this process they grow in size and complexity, and incrementally move away from their initial design. This phenomenon, known as software evolution [1], makes it difficult to maintain a software system, which claims a share estimated up to 90% of total software costs [2], [3], of which 60% is spent in understanding the system [4].

The high cost of software maintenance results from many factors: Documentation is often not updated, or non existent [5]; because of the continuous turnover of developers, changes to a software system are often performed by developers with a limited knowledge of the system. An established technique to deal with these problems and ease software maintenance is reverse engineering. Chikofsky and Cross defined reverse engineering as "the process of analyzing a subject system to (1) identify the systems components and their interrelationships, and to (2) create representations of the system in another form or at a higher level of abstractions" [6].

Most reverse engineering research is mainly concerned with satisfying these goals using different abstraction levels (*e.g.,* code level [7], and design level [8], [9]). The problem with following the above definition is that it takes into account a single software system focusing on either the project or its contributors. However, software systems are rarely developed in isolation, but within a same environment: an abstraction level called ecosystem. A software ecosystem is defined as "a collection of software projects, which are developed and co-evolve together in the same environment" [10].

To analyze software ecosystems at any abstraction level, techniques are required to cope with the huge amount of data available about the evolution process.

Two analysis techniques have been effectively used to convey the results to the end user: metrics [11], [12], [8], and visualization [7], [13], [9]. The advantage of software visualization over pure metrics is that it uses the brain's ability of remembering images [14] and extracting patterns and anomalies from the data that are unlikely when data or numbers are presented in tables or text [15]. Diehl defines software visualization as "the visualization of artifacts related to software and its development process [...] including for example program code, requirements and design documentation, changes to source code, and bug reports" [14].

We present Complicity, a web-based interactive application that visualizes –at different abstraction levels– the data extracted from the Git web interface of super-repositories. Complicity makes use of metrics and visualization to analyze software ecosystems in a bottom-up approach. The user can start analyzing a project or a contributor, and go to the ecosystem level to give a higher-level context for the individual analysis. Using the GNOME super-repository as a case study, we demonstrate the use of Complicity to better understand how an individual project affects the events at ecosystem level, and follow a contributor's involvement in different projects.

The first contribution of this paper is Complicity, a web-based and interactive *application* to visualize software ecosystems. Complicity uses information available in the Git web interface to visualize super-repositories at different abstraction levels. The second contribution is a bottom-up *approach* to study various aspects of a software ecosystem exemplified on a *case study* of the GNOME ecosystem. We demonstrate what information is available with Complicity, and how the visualizations can support software analysts to understand the evolution of projects and contributors within an ecosystem.

II. Related Work

Software evolution visualization research has mostly been targeting single software systems. Seesoft [7] shows the change history of files by mapping one line of code to a pixel line, in which the color of the line represents the recency of its change. Fisheye (http://www.atlassian.com/software/fisheye/), a commercial web-based tool, allows one to visualize the evolution of a single system by looking at charts and matrices of commit activity. The Evolution Matrix [16] displays the evolution of a system at class and system level, with rows mapping different classes and columns mapping versions of the system. Wettel *et al.* [9] exploit the city metaphor to visualize the current state of an object-oriented system and its evolution over different versions.

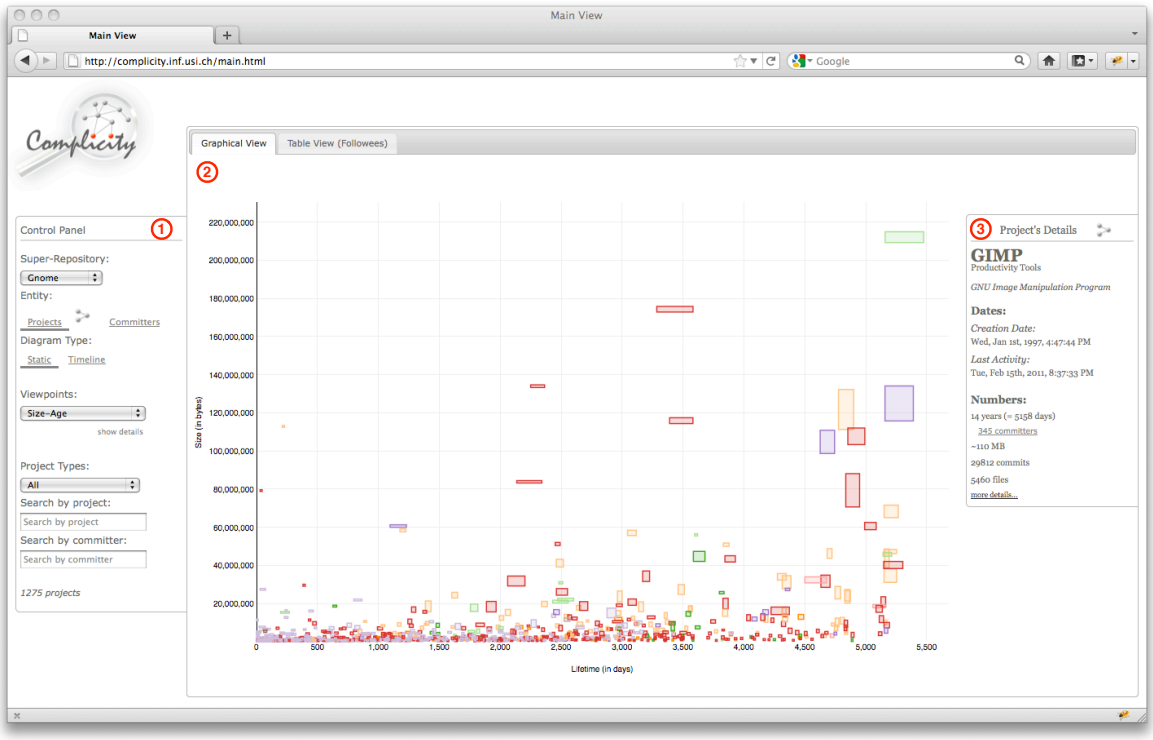

Fig. 1. Main View of Complicity visualizing the GNOME projects at ecosystem level, and the details of the Gimp project

Recent work has analyzed social aspects of a project's evolution (*e.g.,* activity, communication structure, and knowledge flow), which influence the changing process and is crucial to understand the evolution of a software system.

The Ownership Map [13] identifies the owner of every single file within a software system. A file is represented with a line, a disc defines a file change, and the color of the line and the disc defines the owner and the committer, respectively. In Maispion [17], the authors analyzed the activity in the mailing list and version control systems (VCS) of a single project to reveal communication behavior within it. They answered questions such as "Is there a main driver?" and "When are the developers most active?". Oezbek *et al.* [18] checked whether the onion communication model is applicable to open-source systems using mailing lists as data source. They found that the core developers are highly interactive and tightly interconnected.

These approaches use visualization techniques to support the comprehension of a single software system. Complicity uses some of these visualization techniques (*e.g.,* two dimensional boxes to encode different metrics and visualizes contributors' activity) to support the understanding of software ecosystems.

Software ecosystems analysis is an under-researched area. FLOSSMole [19] aims at mining free, libre, and open-source software (FLOSS) super-repositories (*e.g.,* sourceforge) and making general information on these projects publicly available. The data from the VCS or any other data source is not mined. López-Fernández *et al.* apply social network analysis to FLOSS projects, such as KDE, and Apache [12]. They found out that committers of the GNOME and KDE are more tightly connected than the ones of the Apache, because of the GNOME's and KDE's projects technical proximity. Ohloh (http://www.ohloh.net/) is an online directory of FLOSS projects and its developers. It retrieves data from different VCS and uses different metrics (*e.g.,* number of commits, number of lines of code) to provide some visualizations that show various aspects of the projects' evolution. Lungu focuses his work on reverse engineering software ecosystems [10]. He created SPO, an interactive tool that can be used to analyze the evolution of software ecosystems. SPO differentiates between two aspects, project and developer, for which the ecosystem plays one of two different roles: *focus*, to better understand the ecosystem; or *context*, to understand a single entity of the ecosystem. Seichter *et al.* introduced an approach of knowledge management using a social network of software artifacts in which the knowledge is attached to an artifact rather than a contributor [20]. The advantage is that if a developer leaves a software project, the knowledge remains within the project. Goeminne and Mens provide a framework to mine VCS, mailing lists and bug tracking databases, to analyze and visualize mainly the mailing, and commit activity of FLOSS ecosystems [21]. They define an ecosystem as "the source code together with the user and developer communities surrounding the software".

In our work we combine software visualization with software metrics, by focusing not only on a single entity or the ecosystem level, but enabling the user to switch between the ecosystem and entity level, and between project and contributor views.

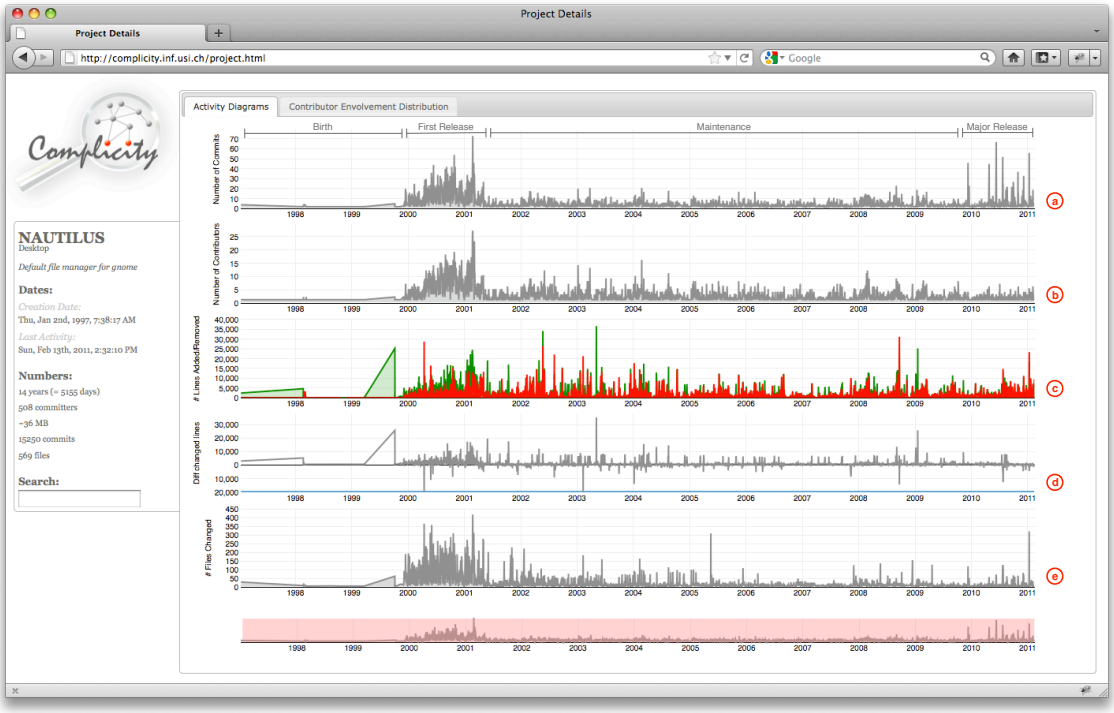

Fig. 2. Project Detail Page (here: Activity per day over the entire lifetime of the *Nautilus* project (a. number of commits, b. number of contributors, c. number of lines added (green) vs. number of lines removed (red), d. difference between number of lines added and removed, e. number of files changed))

III. Complicity

Complicity (http://complicity.inf.usi.ch) is a web-based visualization tool that allows the user to interactively explore, analyze and understand the evolution of super-repositories at two different abstraction levels: entity (single project or contributor), and ecosystem (a group of projects or contributors).

User Interface. We kept the user interface simple to avoid distracting the user from the analysis tasks, whereas the shapes in the visualizations are colored to attract their attention.

The *main page* (see Figure 1) is divided into three main parts: (1) the control panel on the left, (2) the main graphs of the ecosystem in the center of the page, (3) a quick view panel of a project's or contributor's details on the right, which appears by clicking on a shape in the graph.

Control Panel. It gives the user the possibility to (a) analyze different super-repositories, (b) choose between two entity types (project and committer) for the visualization at ecosystem level, (c) navigate through predefined viewpoints or change their settings, and (d) search for projects or contributors of interest by project type, project name or contributor name.

Graphical View. It visualizes the available data for the selected super-repository and the selected entity type as scatter plots. Every box represents either a single project or contributor, depending on the entity type selected, and reflects up to five different metrics: position on x and y axis, width, height and color. By clicking on a shape a detail panel on the right appears.

Entity Details Panel. It provides general information about the selected entity, which goes from name and date of the first and last commit, up to number of commits, number of contributors or projects, *etc.* From this panel the user has the possibility to get more details and further analyze the selected project or contributor in a new window.

The *detail page* of a project or contributor (see Figure 2) has a similar layout as the main page, with the exception that the control panel on the left is replaced with an overview of the selected project's or contributor's details.

In the center of the page, the user can choose between two views: (a) activity diagrams and (b) projects/contributors involvement distribution. The activity diagrams view allows the user to compare the activity of the selected entity in terms of number of commits, number of contributors/projects, number of lines added versus number of lines removed, and number of files changed. In the involvement distribution view, the analyst gets an idea about how many contributors have been involved, when, and for how long, or –in case of a selected contributor– how many projects he worked on and what his speciality is.

Architecture. Figure 3 shows the architecture and the backend of Complicity. To fetch the relevant data from the Git web interface into our database, we developed a number of Java programs. The crawler stores a copy of the web pages from the web interface of the Git super-repositories locally before the parser extracts the data, and stores it in a database. The metrics calculator takes the extracted data, prepares and stores it in such a way that Complicity can easily retrieve the data necessary for the visualization without having to calculate the metrics on the fly.

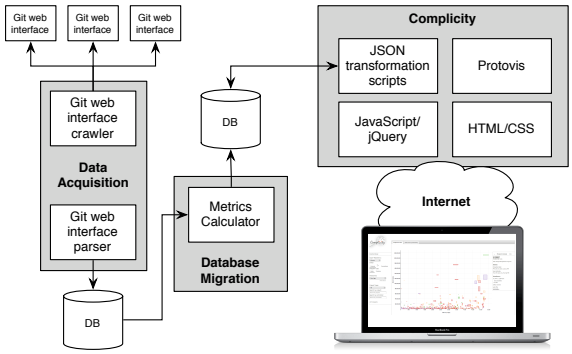

Fig. 3. Architecture of Complicity and the backend

PHP scripts retrieve the data from the database and convert it into JSON objects, the format required for the visualization. The graphical visualization of the data is done by an external toolkit, called Protovis[1]. The user interface of Complicity is solely written in HTML 5, CSS 3 and Javascript using jQuery[2] for user interface components and interactions.

Data Model. Figure 4 illustrates Complicity's data model.

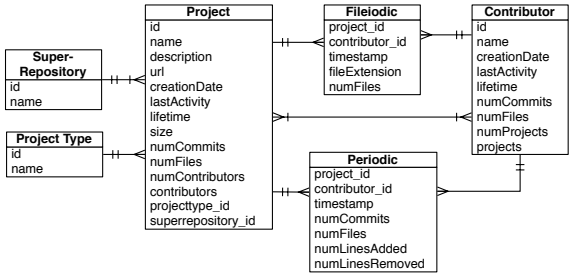

Fig. 4. Data model behind the Complicity visualizations

Each project is attached to a super-repository and to a project type. The project table contains the general information, *e.g.,* name, and description, but also the pre-calculated metrics, such as number of commits (numCommits), number of files (numFiles). The contributor table contains general information as well as pre-calculated metrics. A project can have many contributors and a developer can work on multiple projects.

The tables Periodic and Fileiodic contain information about the changes done to the project. The difference is that Periodic contains the general changes based on the metrics (*e.g.,* number of commits, number of files), whereas Fileiodic contains the data necessary to visualize the specialty of a contributor based on the number of files she has changed with specific extension.

IV. Stories of the GNOME Project

GNOME is a desktop environment for GNU/Linux/Unix composed of many free and open-source software systems. It was created in 1997 by two students, and since then it has grown in popularity. We show how Complicity can support software analysis at ecosystem level. Using a bottom-up approach, we first analyze a single project regarding its activity and community support before moving to the ecosystem level and trying to reveal some patterns in the contributors' affection to either translation or development work. In a second example, we analyze a contributor's activity and expertise before examining at ecosystem level how he affected the GNOME project.

This approach is of interest to analysts who want to understand the evolution process of a project and its impact on GNOME, or to analyze the role of contributors on individual projects and at ecosystem level. Several maintenance needs may lead analysts to use visual exploration of software ecosystems, *e.g.,* to identify critical parts of the system to prioritize maintenance; identify the main contributors –who retain the knowledge– of a project; and to identify dependencies among projects to be aware of ripple effect of a project's release on depending projects. The approach is also of interest to developers who want to get involved in the projects, and need to understand their dynamics before becoming active contributors.

We focus on Nautilus, GNOME's default file manager.

A. The Nautilus project and its impact on the ecosystem

Project Activity Diagrams are a good starting point to get a first idea of Nautilus' evolution beyond the basic information available in the project's details. They illustrate the project's daily activity comparing six different metrics: number of commits, number of contributors, number of lines added and removed, difference between number of lines added and removed, and number of files changed. We use an area chart for all activity metrics, except the line-difference metric, which is drawn as a line chart. All of them have the time on the x-axis and one of the six metrics on the y-axis. The number of lines added (in green) and removed (in red) is drawn in a same diagram for better comparison. These diagrams allow us to identify different phases of a project. In Figure 2 we observe that Nautilus was created in 1997 (birth), when also GNOME was created. Between 2000/2001 its popularity increased in terms of contributors, in number of commits per day and in number of lines added/removed (first major release). Afterwards the number of commits and contributors decreased and stabilized. Also the number of lines added/removed equalized (maintenance). In 2010, the number of commits increased again until Feb. 2011 (new major release).

Beside the different phases, we observe that there is continuous development taking place almost every day, which is an indicator for an active development team. The next step is to get familiar with its contributors: Who worked, when, and for how long on the project? This information is presented by the *contributors involvement distribution* view in Figure 5.

Contributors Involvement. This view gives information about the number of commits, but it does not allow a comparison of different metrics at the same time. Instead, it provides a more detailed view on the people who contributed to the project over time, so that the drivers of the different phases can be identified. This information is crucial to understand the flow of knowledge (Is there a center of power? Are there smooth

[1]See http://vis.stanford.edu/protovis/.
[2]See http://jquery.com

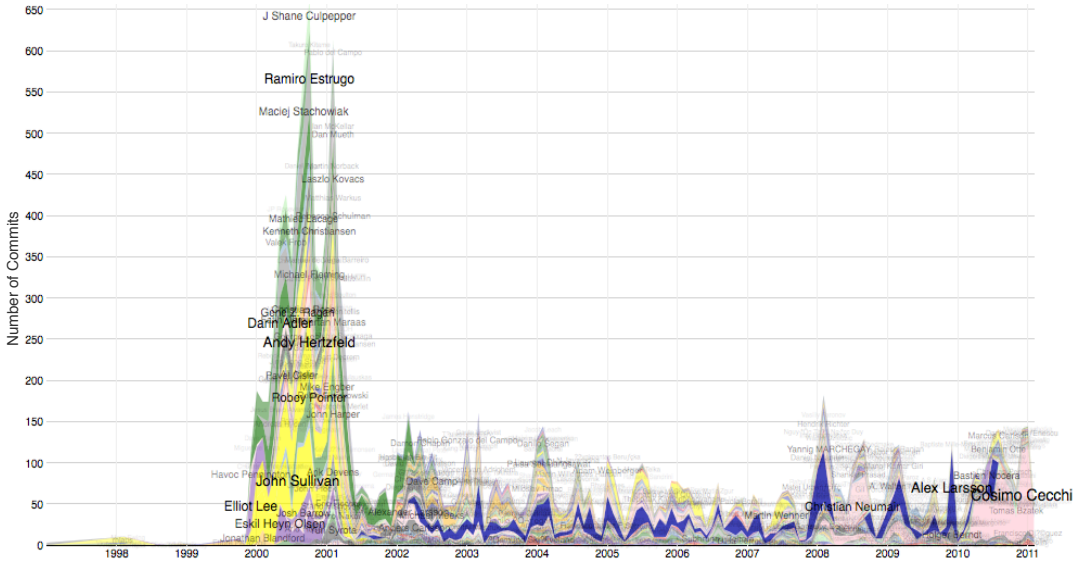

Fig. 5. Contributors Involvement View of the Nautilus project

takeovers?), which is a good basis for a project to evolve well and have a long lifetime. For this visualization, we use a stacked area chart using colors to identify different contributors. The size, the density and the position of the contributor's name are based on the maximum number of commits a person has done at once. Clicking on a single contributor hides all the others, so that the contribution timeline of a single person can be better analyzed. This view shows the evolution on a monthly basis. In Nautilus' first major release phase (see Figure 5), Ramiro Estrugo (gray), as well as Andy Heitzfeld (yellow), Darin Adler (green), and John Sullivan (yellow) are the main initiators of the project. Darin Adler remains the main driver at the beginning of the maintenance phase until the beginning of 2002, when Alex Larsson (blue) takes over. He remained the driver for almost the whole maintenance period. Only by 2008 a new actor enters the scene, Cosimo Cecchi (light pink), who takes over the project. The above diagrams contain information about a single project, how it and its community evolve over time. As Nautilus is only one software system of the GNOME project, it is important to analyze how it impacts the ecosystem.

Affectional Bond - Ecosystem Diagram. This view shows the contributor distribution at ecosystem level, with number of commits on the x-axis, and number of project on the y-axis and as color metric. The width and height of each shape is defined by a contributor's lifetime in days. This view can be used to visualize all contributors of the entire ecosystem or filters can be applied to show only the contributors of a specific project.

This visualization presents a piece of information that cannot be revealed at project level from any of the two views described above: the contributors' general affectional bond to either development or translation work. In this graph the contributors are split into these two groups under the assumption that people who contributed a lot but only to a relatively small number of projects are likely to be developers. Conversely, people who committed less often but to more projects are likely to be translators. This results into the following distribution: people located under a logarithmic-like curve are defined as developers and the ones placed above an exponential-like curve are considered to be translators.

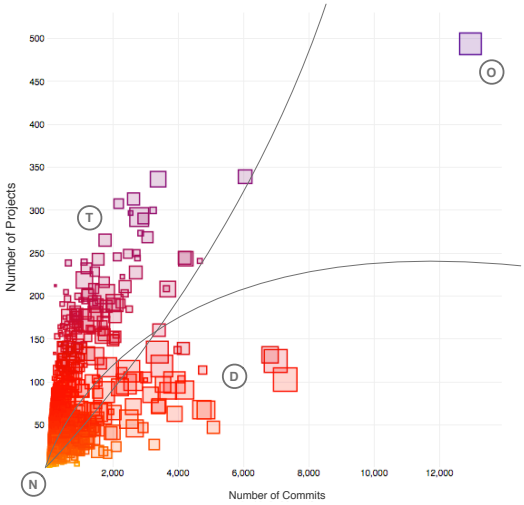

Fig. 6. Affectional Bond view at ecosystem level. Each square represents a committer, where the x position maps the number of commits, the y position and the color the number of projects, and the size the lifetime in number of days (T: translators, D: developers, O: outlier, N: no man's land)

Figure 6 illustrates the contributor distribution of the Nautilus project. It reveals that many people have been working on the project over time and that the distribution between developers (marked as D) and translators (marked as T) is roughly equalized. The contributors located within the high-density bottom left part (marked as N) might not be clearly differentiable. It seems that developers need a longer lifetime

(larger boxes) in order to be clearly differentiable from the translators, who have a more varied lifetime (collection of large and small boxes). The outlier on the top right (marked as O) did a huge amount of changes (positioned to the right) and contributed to many projects (positioned to the top) over a long lifetime (relatively large box). O is Kjartan Maraas, both a developer and a translator within the GNOME project.

B. Impact of the contributor Kjartan Maraas on the ecosystem

As a second example we have chosen Kjartan Maraas as the contributor for the bottom-up analysis. Since 1998 he has been working on 499 different projects, having done more than 15'000 changes by February 2011. During 4,689 days (*ca.* 13 years) he had an average rate of three commits per day.

We know when a contributor did his first and last commit and we can calculate his average commit rate. However it is not possible to know whether he was active all the time or there have been some gaps of inactivity during his lifetime. This information can be extracted from developer activity diagrams.

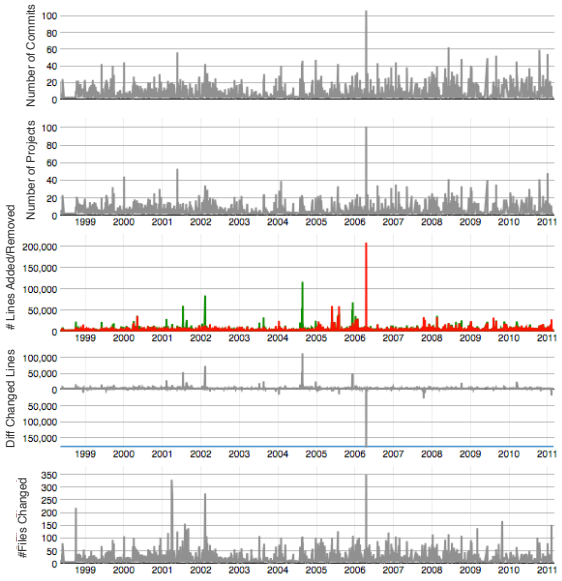

Fig. 7. Activity Diagrams of Kjartan Maraas

Developer Activity Diagrams give an overview of the contributor's daily activity within an ecosystem. They are constructed the same way as the activity diagrams at project level with the only difference that the metric number of contributors is replaced by number of projects. In Figure 7 we observe that Kjartan Maraas has been continuously active with only one exception of few months. Particularly interesting in his past activities is a short period of time (one day), in 2006, during which he added or removed up to 200'000 lines, performed about 100 commits and changed up to 350 files. A piece of information missing in this view is on which projects Kjartan Maraas did these changes. To this aim, we have introduced the project involvement view, presented next.

Project Involvement. This view provides the projects to which a person committed monthly. As for the contributors involvement view, we use a stacked area chart with number of commits on the y-axis and time on the x-axis. Figure 8 shows that Kjartan Maraas has been active in many different projects.

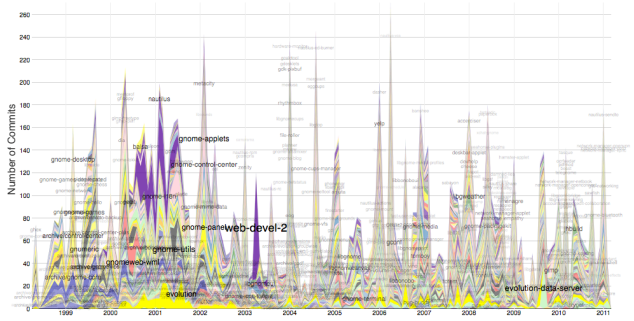

Fig. 8. Projects Involvement View of the Kjartan Maraas

The number of projects he worked on increased after the first few years, which can be revealed by the fact that the colors representing the projects in this view become less differentiable towards the right part of the visualization. Also the color intensity and size of the labels become more similar, which means that he committed with similar rate to these projects. At this point of our analysis it might be interesting to understand how it is possible for a single person to do so many commits and changes while working on so many projects. To answer this question we devised the expertise view.

Expertise View. This visualization yields information about a contributor's expertise based on file extensions. It is drawn as a stacked area chart, counting the number of files that has been changed within a month aggregated by their file extension.

Figure 9 shows the expertise view applied to Kjartan Maraas. He is a translator and C developer, as he changed a large number of .po files (light green) as well as many C files (blue).

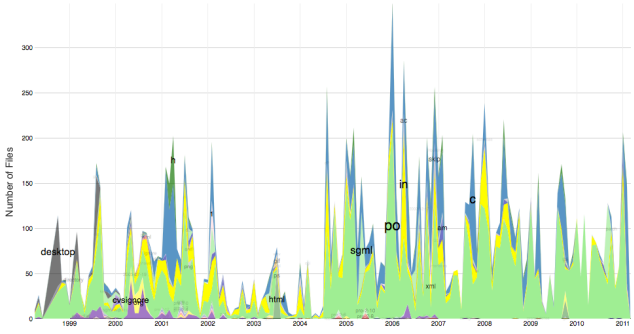

Fig. 9. Expertise View of Kjartan Maraas

The combination of the three previous visualizations explains how Kjartan Maraas does so many changes in so many projects: He regularly worked on the GNOME ecosystem over a very long period. Exploring the projects involvement view, we see that he commits to most of the projects more than once but with a low monthly commit rate.

Knowing that besides being a translator, which makes it possible to work on many projects simultaneously, he is also a developer, we can conclude that he is likely to be a maintainer trying to fix bugs on different projects. Indeed, by checking his profile on LinkedIn[3] we found out that he has been member of the Bugquads at GNOME for almost ten years beside being a member of the release team and of the GNOME foundation.

The views at contributor level provide details about a single person's activity level and expertise. However, to better understand how a contributor affects the GNOME ecosystem, we need visualizations at ecosystem level.

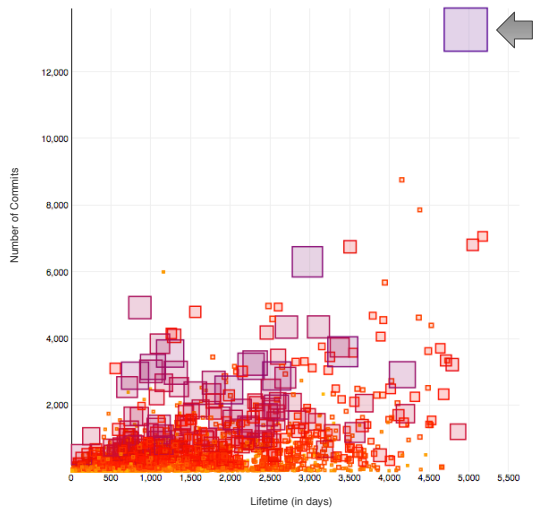

Fig. 10. Kjartan Maraas (marked by the arrow) compared to all other contributors of the GNOME project within the Activity Fire View

The **Activity Fire - Ecosystem Graph** is constructed with number of commits on the y-axis, lifetime on the x-axis, and number of projects as width, height and color of the boxes. It illustrates the distribution of the contributors according to their activity over their lifetime in the GNOME project.

Kjartan Maraas is an outlier compared to the contributors of Nautilus (see Figure 6) and within the GNOME ecosystem (marked by an arrow in Figure 10). He performed by far most commits (positioned to the top) and worked on more projects than anybody else (large box) in the ecosystem. This might be related to his long lifetime (positioned to the right) at GNOME but also to the fact that he is both a translator and a developer.

Projects do not always have long lifetimes. As consequence, Kjartan Maraas worked on many projects but probably not on all at the same time. The following view provides clarification by illustrating the projects' lifetime.

Projects' Lifetime View - Ecosystem Graph. This view shows the projects' life duration by distributing the projects according to their first commit (x-axis), *i.e.,* the creation date, and their last commit (y-axis), which might correspond to the death date if a project has not been changed for a longer period. We define a project as dead if it has not been changed for over a year. Since projects cannot die before they are created, the boxes are distributed over a triangular surface with the first created projects on the left most vertical line, the still active projects on the top most horizontal line, and the short-lived (less than one year) projects on the diagonal line, closing the triangle. The width and height of each box are defined by the total number of commits and the total number of contributors, respectively. The color represents the different project categories and might reveal some trends within the ecosystem or preferences of a single contributor.

[3]http://www.linkedin.com/in/kjartanmaraas

Fig. 11. Projects' Lifetime View showing only the projects Kjartan Maraas contributed to with A: active projects, P: prototypes, and D: dead projects (Color scheme: projects of category archived (red), others (green), desktop (orange), bindings (light blue), development tools (pink), and platform (purple))

Figure 11 shows that many of the projects Kjartan has contributed to have died (D). Most of them are colored red (archived), green (others), or orange (desktop). The projects within this area placed at the diagonal line (P) are likely to be prototypes or projects that have been integrated into other projects. They have a very short lifetime and die almost as soon as they are created. Only the projects on top (A) are still alive, and actively under change. Maraas never worked on all projects at the same time as some of them died before others had been created. Summarizing, we can say that Kjartan Maraas is a very active person with a long lifetime as translator and developer with an affinity for bug fixing. He is the most important person according to the number of commits and the number of projects he has been involved over his lifetime. He is an outstanding person not only compared to the contributors of the Nautilus project but within the entire GNOME ecosystem.

For the Nautilus project, we can conclude that it is under continuous change with active contributors, who are equally distributed number of translators versus developers. In addition, it has at least one main driver in each phase with smooth take-overs, which are essential for a good knowledge flow.

V. Discussion

Importance of different abstraction levels. Our case study shows the importance of different abstraction levels –ecosystem and entity level– and the ability to interactively move from one level to another, as they complete each other. These abstraction levels allow analysts to identify dependencies between projects, contributors, and between both. Also, by integrating both of them in one analysis task, one can illustrate the impact of a single project or contributor on the entire ecosystem.

Versatility of Complicity. We illustrated how one can make use of a bottom-up approach for analyzing the evolution of a software ecosystem. However, Complicity supports both bottom-up and top-down approaches to analyze the evolution of software systems individually and at ecosystem level. Also, we presented multiple views that allowed us to explore the history of a single project and contributor and their impact on software ecosystem, *e.g.,* GNOME. All of the views considered in the case study are implemented in Complicity. Given the space limitations, we did not present all of Complicity's views.

Drawback of basic metrics. We showed that abundant information can be extracted by applying only basic metrics, *e.g.,* number of commits, number of projects, *etc.* but it has to be considered that these metrics might be misleading and should be considered carefully. For instance, the fact one person commits more than another does not necessarily mean that the former does more work than the latter. Instead, it depends on the committer's attitude on performing large or small commits.

Git committers versus authors. Compared to other version control systems, Git differentiates between author and committer[4]: The author is the person who originally wrote the work, and the committer is the person who last applied the work. We ignore the committer. This might lead to wrong conclusions if the committers differ from the authors most of the time.

VI. Conclusion

We presented Complicity, a web-based visualization tool that allows interactive exploration and analysis of software ecosystems evolution. The data used on the analysis has been collected by reverse engineering super-repositories. Complicity is applicable to analyze software ecosystems in a bottom-up approach. In addition, we analyzed the history of a project and a contributor, and showed their impacts on the ecosystem using predefined views at two different abstraction levels. However, analysts can use Complicity to visually explore the provided data on their own using other viewpoints and approaches.

Future work. First, we intend to explore other ecosystems with complicity and compare the results. Second, we plan to improve our technique to automatically eliminate duplicate contributors. This is a common problem, known as aliasing [22], as contributors use different email addresses, and different names to commit to a repository. Different solutions have been proposed to tackle this problem, such as using the Levenshtein distance [17] or fuzzy string similarity, domain name matching, clustering, and heuristics [22]. To eliminate duplicate contributors, we have focused on aggregating people based on their email addresses, names and using the Levenshtein distance. Identifying people based on their commit rate and the projects they worked on would help to identify and aggregate contributors with different names and email addresses. Lastly, we aim at incorporating other metrics and data from other types of archives (*e.g.,* email archives, bug trackers and forums).

[4]See http://progit.org/book/ch2-3.html

Acknowledgments. Hattori is supported by the Swiss Science foundation (SNF Project No. 129496, "GSync").

References

[1] M. M. Lehman, "Programs, life cycles, and laws of software evolution," in *Proceedings of the IEEE*, vol. 68, no. 9, September 1980, pp. 1060–1076.
[2] B. P. Lientz and E. B. Swanson, "Problems in application software maintenance," *Commun. ACM*, vol. 24, pp. 763–769, November 1981.
[3] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, 2000.
[4] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, 1989.
[5] R. Kazman and S. J. Carrière, "View extraction and view fusion in architectural understanding," in *Proceedings of ICSR 1998*. IEEE, 1998, pp. 290–299.
[6] E. Chikofsky and J. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, Jan. 1990.
[7] S. G. Eick, J. L. Steffen, and E. E. Sumner, "Seesoft - a tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, pp. 957–968, 1992.
[8] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger, "Codecrawler — an information visualization tool for program comprehension," in *Proceedings of ICSE 2005 (27th IEEE International Conference on Software Engineering)*. ACM Press, 2005, pp. 672–673.
[9] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Proceedings of VISSOFT 2007*. IEEE CS Press, 2007, pp. 92–99.
[10] M. Lungu, "Reverse engineering software ecosystems," Ph.D. dissertation, University of Lugano, Switzerland, Oct. 2009.
[11] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.
[12] L. López-Fernández, G. Robles, J. M. Gonzales-Barahona, and I. Herraiz, "Applying social network analysis techniques to community-driven libre software projects," *International Journal of Information Technology and Web Engineering*, vol. 1, no. 3, pp. 27–48, July-September 2006.
[13] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Proceedings of IWPSE 2005*. IEEE, 2005, pp. 113–122.
[14] S. Diehl, *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Verlag, Berlin, April 2007.
[15] B. A. Price, I. S. Small, and R. M. Baecker, "A taxonomy of software visualization," in *Proceedings of the 25th Hawaii International Conference on System Sciences*, vol. 2, Kauai, HI, USA, January 1992, pp. 597–606.
[16] M. Lanza, "The evolution matrix: Recovering software evolution using software visualization techniques," in *Proceedings of IWPSE 2001*. ACM Press, 2001, pp. 37–42.
[17] F. Stephany, T. Mens, and T. Gîrba, "Maispion: A tool for analysing and visualising open source software developer communities," in *Proceedings of IWST 2009*. ACM Press, 2009, pp. 50–57.
[18] C. Oezbek, L. Prechelt, and F. Thiel, "The onion has cancer: Some social network analysis visualizations of open source project communication," in *Proceedings of FLOSS 2010*. ACM Press, 2010, pp. 5–10.
[19] M. Conklin, J. Howison, and K. Crowston, "Collaboration using ossmole: a repository of floss data analyses," in *SIGSOFT Softw. Eng. Notes*, 2005.
[20] D. Seichter, D. Dhungana, A. Pleuss, and B. Hauptmann, "Knowledge management in software ecosystems: software artefacts as first-class citizens," in *Proceedings of the Fourth European Conference on Software Architecture*, ser. ECSA '10. ACM, 2010, pp. 119–126.
[21] M. Goeminne and T. Mens, "A framework for analysing and visualising open source software ecosystems," in *Proceedings of IWPSE-EVOL 2010*. ACM Press, 2010, pp. 42–47.
[22] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR 2006)*. ACM, 2006, pp. 137–143.