Software Visualization Tool for Evaluating API Usage in the Context of Software Ecosystems: A Proof of Concept

Rodrigo Avancini[1,2(✉)], Fábio Fagundes Silveira[1], Eduardo Martins Guerra[3], and Pedro R. Andrade[2]

[1] Federal University of São Paulo – UNIFESP, São José dos Campos, Brazil
avancinirodrigo@gmail.com, fsilveira@unifesp.br
[2] National Institute for Space Research – INPE, São José dos Campos, Brazil
pedro.andrade@inpe.br
[3] Free University of Bolzen-Bolzano – UNIBZ, Bozen-Bolzano, Italy
guerraem@gmail.com

Abstract. Software Ecosystem (SECO) is a consolidated research area in software engineering, emerging as a paradigm for understanding dynamics and relationships among software systems that collaborate with each other to achieve their goals. Understanding the ecosystem and how its elements interact is essential for software evolution, especially for those that provide functions and services for other systems, such as software APIs. Once an API is being used by different software, future changes need to be made in a systematic and appropriate manner, considering the whole ecosystem. However, there is a lack of formal and effective ways for APIs evaluation in the context of SECO. Thus, in this paper, we present Ecolyzer, a prototype tool that aims to support the analysis of API usage considering its ecosystem through interactive visualization. To demonstrate the feasibility of our tool, we conducted a proof of concept (PoC) using an open-source platform API. The results obtained with Ecolyzer are useful and show that the prototype meets the goals described for the accomplishment of this work.

Keywords: Software ecosystems · Software visualization · API usage

1 Introduction

Increasingly, software systems depend on other software to leverage their business. This fact occurs due to the vast availability of specialized software and components that have already established themselves in their respective domains. There are rare cases in which software is developed from scratch without reuse [25]. Nevertheless, the vast majority of analysis and development support tools do not consider the relationships among the systems and their software dependencies, having a limited view of the impacts caused by the inevitable software evolution [9].

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12254, pp. 335–350, 2020.
https://doi.org/10.1007/978-3-030-58817-5_26

From the perspective of software that provides functions and services, guaranteeing permanence in the market does not depend only on its correct internal functioning, but also on a constant balance among software that use it [24]. This concern led to the study of software systems in a broader context, involving everyone that somehow influence their survival.

To investigate the relationship among software products and their dependencies, the term software ecosystems (SECO) emerged in 2003 [3], referring to a set of software that has some degree of symbiosis among them [20]. Several studies have emerged to model and represent a software ecosystem [21]. Despite the advances achieved so far, there is still no consensus between both academia and industry on the best way to present software ecosystem entities [22]. Also, the lack of formalism and tools to support the modeling and visualization of SECO is hampering the advance of knowledge in the area [15].

An API is a particular type of software that is intrinsically designed to deal with ecosystems. Useful APIs increase the productivity of developers who use them, the quality of the software they produce, as well as the corporate bottom line [5]. However, its evolution represents a high risk for its users since its changes might propagate other changes in the entire ecosystem, requiring additional maintenance by the software that uses the API. In the worst case scenario, the entire API is replaced by a more satisfactory solution. Thus, the evolution of an API needs to be strategically geared toward meeting the goals of its users. Unfortunately, there is a lack of effective ways to analyze APIs considering their ecosystem, which makes it difficult to devise a better design strategy for their evolution [11].

This article presents *Ecolyzer*, a tool that aims to explore the dependencies of software components considering its ecosystem. The novelty of our approach consists of the employment of visualizations, making it possible to analyze and explore an API that provides functions and services for other software, taking into account all software that use it, thus contributing to its development and evolution. To achieve our goals, we developed a tool that gathers the main ideas proposed in the literature on the visualization of software ecosystems. We adapted them to evaluate the API usage by other software through a proof of concept by using a real software.

The remainder of this paper is structured as follows: Sect. 2 presents the background notions related to software ecosystems and an overview of current work involving visualization tools. Section 3 describes our approach for analyzing API usage, considering its ecosystem through an interactive visualization tool. Section 4 presents a proof of concept to demonstrate the feasibility of our approach. Finally, Sect. 5 presents our final remarks and plans for future works.

2 Software Ecosystems and Tools

There are many definitions related to software ecosystems. Some of them are more focused on technical aspects of software engineering and software development, while others are more focused on business, encompassing the software

product and services, organizations, markets, vendors, and all actors involved [14,21]. Although both distinctions are important, in this article, we limit our review to solutions available in the literature of software engineering applied to ecosystems.

2.1 Software Ecosystem

The term ecosystem was originated from ecology, which in short consists of a set of actors, connected to other actors with connections among them, interacting as a system [6]. In the context of software ecosystems, actors are represented by software systems, and the connections are the dependencies among them.

According to Barbosa and Alves [3], the term software ecosystem appeared in 2003, referring to a set of software that has some degree of symbiosis among them [20]. In turn, Lungu [17] defined it as a collection of software projects that belong to an organization and are developed in parallel with it. Later, this definition was refined for a collection of software projects that are developed and evolve together in the same environment [18]. As reported by Goeminne and Mens [9], SECO consists of the source code together with the communities of users and developers that surround a software. In consonance to Bosch [6], SECO consists of a central platform with products and applications built using this platform, developed by external developers.

These definitions of SECO seem more appropriate when it comes to the software engineering area. They were merged or reformulated as new work emerged, always referring to a collection of software projects, which: (1) evolve together and are maintained by the same developer community [22], (2) are dependent on each other linked through project artifacts, common contributors, and support communities [25], and (3) base their relationships on a common interest in a central software technology [12,21].

Despite the considerable number of definitions, the essence of software ecosystems remains the same as that of ecology: a set of actors and their relationships, where actors are represented by software products, developers, and users communities, and the relationships are the links and dependencies among the actors.

In this work, we define software ecosystem, from the point of view of software engineering, as a set of software projects, which use a common (or central) software platform that provides resources for other software, through functions and services. These functions and services may be symbiosis or not, disregarding actors and relationships that can not be obtained in a computationally way, that is, restricted to systems. This definition allows us to study an API considering the entire ecosystem, using a single tool that extracts data, analyzes and presents the central software, all in an automated way, helping in its maintenance, improvements, evolution, and insights about the system, besides helping the evolution in the area of SECO.

2.2 Ecosystem Visualization Tools

Despite several studies exploring the characteristics and relationships among software ecosystem entities, there are still few approaches dealing with software ecosystem analysis and visualization tools. This situation is even worse when it comes to automatized tools that lead the entire process from data extraction and analysis to presentation.

Basically, a software ecosystem at a lower level is comprised of systems and their relationships. From a business perspective, it can involve internal and external actors, organizations, companies, and other stakeholders [21]. Given these elements, models are created, showing how they are interconnected and how these connections are. Therefore, the relationship is an essential entity in an ecosystem, and it can mean dependencies among components, trade relationships, and collaborations [15].

There are benefits provided by the modeling and visualization of a software ecosystem, such as a way to understand it, enabling analysis, formalization, and providing insights about it [15]. These advantages contribute technically not only at the software level, in its maintenance, improvements, and evolution, but also at the project level, helping in economic and strategic decision-making process [4]. However, most of the SECO visualization tools currently proposed to outline the relationships among the elements of the ecosystem are composed of tailored tools designed for one or a few scientific experiments [9]. Thus, software engineering becomes fundamental to enable the advance in this area [8,19].

Lungu [18] can be considered one of the pioneers in the development of tools that support the analysis of software ecosystems, enabling visualization and exploration of an ecosystem through an interactive web interface, named Small Project Observatory (SPO). Developed as a web application, SPO extracts information about the ecosystem from data available in the versioning repositories of the projects. The authors call those repositories of an ecosystem as the super-repository. The architecture of SPO is depicted in Fig. 1.

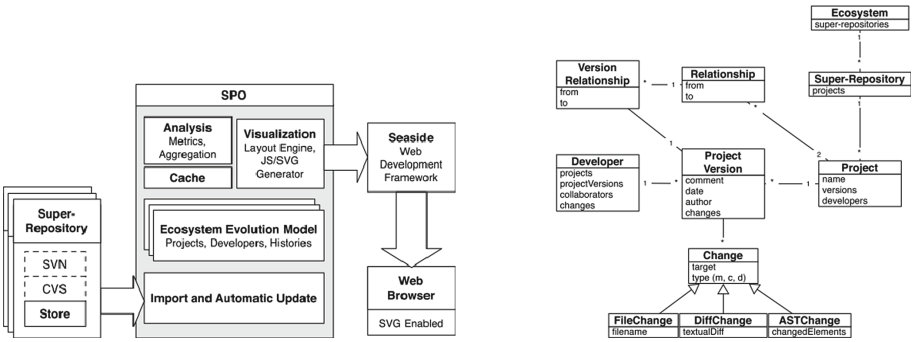
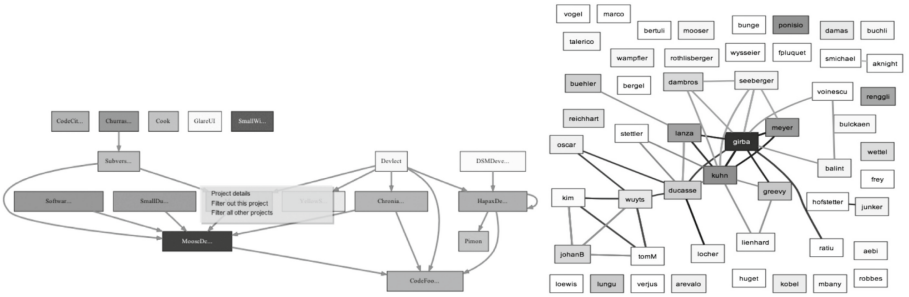


Fig. 1. The architecture and the ecosystem meta-model of SPO [18].



(a) Projects dependencies view. (b) Developers collaboration view.

Fig. 2. Views of SPO [18].

Regarding visualization, SPO presents information taking into account all the super-repository, such as projects, developers, and inter-project relationships. The views are presented in Fig. 2.

Also, in their work, Lungu [18] present a usability study of the tool, showing that the main problems regard to the slowness of the system and the lack of scalability when many components are presented. It was pointed out the low interactivity on the visual elements, the lack of support for users to create their own views, and the impossibility for navigation to check relationships at a lower level, such as calls to methods between software.

Although SPO has a simple architecture and limited views and resources, their work shows elements and concerns to be considered when building a tool for analyzing and visualizing software ecosystems. Here, it can be cited the data extraction from version control systems, the statistical analysis of the code, statistics and metrics, and the representation of relevant elements of the ecosystem at lower level [18].

Following a slightly different approach, the tool presented by Goeminne and Mens [9] concerns not only with the relationships that can be extracted from the repositories but also from other data sources, such as mailing lists and bug tracking. For this, they created a generic multi-layer framework that is divided into three layers in which each layer processes and makes information available to the upper layers from the data sources to the application layer. The lowest layer is called the mining layer, which extracts information according to the data source. The middle layer is called the analysis layer, which extracts metrics under the data extracted from the mining layer. Finally, the upper one is called the application layer, which is responsible for presenting the data.

Goeminne and Mens [9] provide a resource to evaluate metrics commit to commit, making it possible to analyze in a straightforward way the evolution of the ecosystem. Although the tool proposed by the authors was designed to help understanding the evolution of software ecosystems by providing visualization of metrics through charts, it does not present ways of representing the relationships and dependencies among the systems.

Another important work presents a software ecosystem visualization and analysis dashboard tool named SECONDA [22]. It is similar to the Goeminne and Mens's [9] framework and follows the same approach regarding architecture, having modules for reporting, visualization, static analysis, metrics computing, and identity matching. In addition, the tool offers a dashboard with a variety of charts with metrics and statistics, both for projects and developers individually, as well as for analyzing them together. Despite that, SECONDA has not improved the representation of entities and their relationships within the software ecosystem, as its visualization presents only charts with metrics.

The work described by Santana and Werner [25] depicts a software ecosystem by visualizing the relationships of related software projects, providing views of relationships among project members, and relationships at the component level of the systems, such as method calls and object declaration between the systems. For that purpose, a procedure that consists of data extracting, data processing, and SECO visualization was built.

Other approaches have also been proposed in the literature for SECO modeling and visualization, but either they are not considered specific tools for data extraction and visualization or do not present the ecosystem entities and their relationships. Among the modeling techniques proposed to visualize software ecosystems, the most common are: ad hoc notations, tabular representations, conceptual maps, class diagrams, network graphs, software supply network (SSN), meta-models, and i* modeling [21].

Different from the solutions proposed by other researchers related to the visualization of ecosystems, our proposal focuses on a central system as the main software. We explore its components not in isolation as usual, but considering the software that use them.

3 Software Visualization Approach for API Usage

The approaches proposed in previous works to create modeling and visualization of software ecosystems are powerful and supportive of SECO analysis, but as noted by Jansen et al. [15], there are still opportunities for enhancements. This section presents a tool for analyzing central software, such as an application programming interface (API), a software development kit (SDK), a library, or any software that provides services to other software, considering its ecosystem. From now on, the term API always refers to the central system of our approach.

3.1 Overview

Common requirements need to be observed in the software ecosystem visualization tools, such as concerns about system modularity, data extraction in version control systems, and other data sources such as bug tracking and mailing list. Besides, features as static analysis, calculation of statistics and metrics, as well as the presentation of the ecosystem – its actors and relationships – must

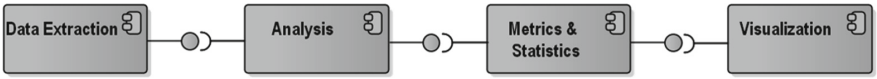


Fig. 3. Components of a SECO visualization tool.

be considered. Thus, a generic SECO visualization tool consists of component architecture, as shown in Fig. 3.

The first step in a SECO visualization system is related to data extraction from some data source. With the extracted data in hand, the second step consists of analyzing such data, which, in the context of SECO, refers to identify the relationships among the involved systems. The third step is responsible for extracting metrics and statistics from the mapped ecosystem. In this step, metrics and statistics should consider all software and their relationships within the ecosystem, and not only a software in isolation as in traditional approaches [16,25].

The novelty of our approach consists in SECO visualization that different from other works in which the concern is to show the entire ecosystem and their relationships, our tool provides API-centric visualization and all its components, taking into account all the systems that use this API. The advantage of this visualization is to enable the analysis in-depth of the API quickly and intuitively, as shown in the following sections.

3.2 Data Extraction, Analysis, and Metrics

Although it seems complex at first [25], data extraction can be done by software repository mining (MSR) techniques, which makes the task feasible. There are currently some MSR APIs available depending on the type of data source [26]. Among data that are possible to obtain from repositories, we highlight the source code and metadata of commits, such as developers, changes in the source code, date of the changes, among others. The metadata of commits makes it possible to study the evolution of systems [9]. Besides, data can be extracted directly from the software's directory structure regardless of a version control system. This data is enough to identify relationships among the software.

In the analysis process, relationships are created from the elements of the source code using functions (or methods) or some other resource made available by the components of the central API and calls (or invoking) for these functions coming from other software. The feasibility of this step depends on the source code analysis APIs, which build an abstract syntax tree (AST), providing the code elements, allowing the static analysis of the source code. The challenge of the analysis consists in identifying, for each call of a function from other software, whether it belongs to some component of the central API. Moreover, the relationships can contain information about data provided by version control systems such as authors, code elements creation, and modification dates.

After the analysis step, some metrics and statistics in the context of SECO can be determined, such as: (1) the most-used components, the least used ones,

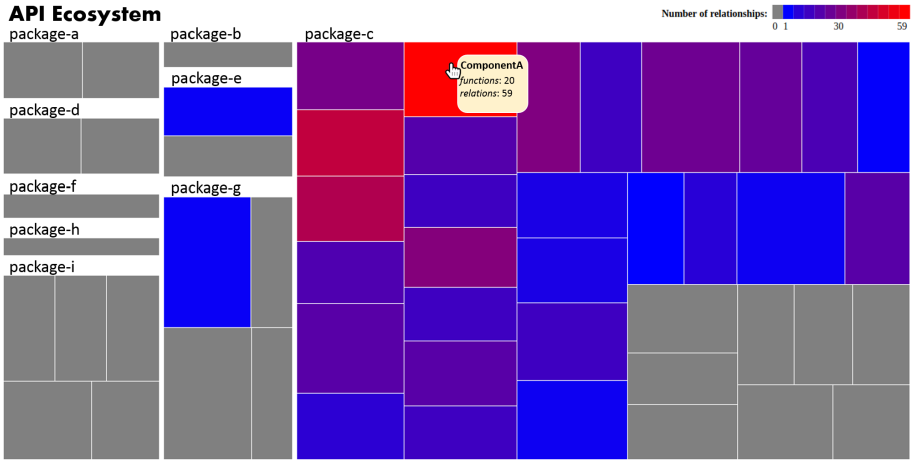


Fig. 4. Central system view of Ecolyzer.

and those not used by other software; (2) the software that use a specific API component; (3) the components of other software that use a specific API component; (4) which API functions are used by other software; (5) the number of functions used by other software over the number of functions available within an API component; and (6) the number of different functions used by a component of another software over the total number of functions used by the component. These metrics and statistics are better explained through our proposed visualization approach.

3.3 Software Ecosystem Visualization

Our approach provides a way to analyze an API as a whole, not only in an isolated way. The visualization considers other software and the relationships among its components at the level of elements of the source code, such as functions (or methods), function calls, and other shared resources. For this purpose, the visualization consists of a treemap combined with a heatmap. This strategy provides more dimensions for analysis through visual elements. Figure 4 depicts an example of visualizing an API.

In this view, each component is a rectangle grouped by system modules or packages. The area of each rectangle represents the number of operations provided by the given component. The colors inform the number of relationships that the respective component has with other software. The legend on upper right describes the number of relationships. Also, the rectangles have a tooltip for checking the component name and other additional information relevant to SECO.

By selecting an API component, it is possible to view the software that use this component and the software components that have some relationship with the selected API component. As can be seen in Fig. 5, this view differs slightly

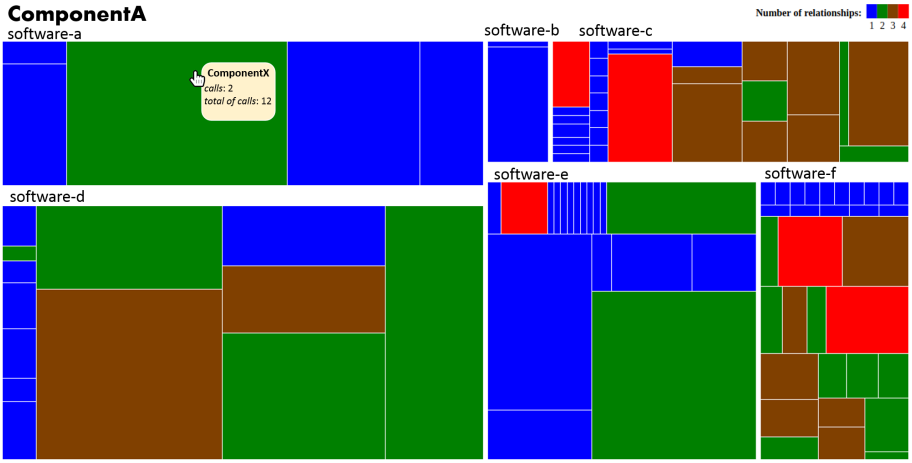


Fig. 5. Component view of Ecolyzer.

from the API view (Fig. 4). The areas of the rectangles represent the total number of calls to operations of some API component, and the colors represent the number of different called operations, disregarding calls to the same operation more than once.

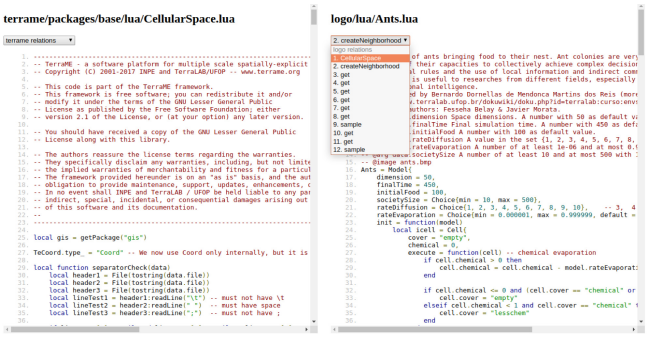
If it is necessary to go deeper into the source code, selecting a software component makes it possible to view the code of the API component, side by side, with the code of the component that uses it. Also, this view provides a way to navigate among components by selecting common code elements between the source codes, as shown in Fig. 6.

3.4 The Ecolyzer

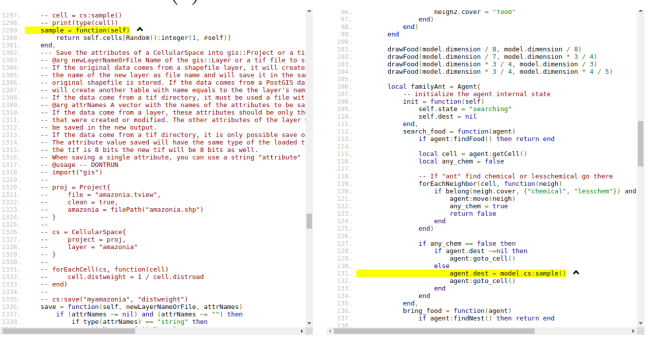
Aiming to achieve our goals, we developed a multi-tiered software system for data extraction, analysis, and visualization of the API's components, considering its ecosystem, named Ecolyzer. Its architecture, along with the main modules, is presented in Fig. 7.

Ecolyzer was developed as a client-server application and its architecture is based on RESTful web services. The main motivations that led to this architectural decision were simplicity, popularity, and the ease of developing RESTful applications due to the available frameworks [23]. The layers of Ecolyzer are briefly described here since it is beyond the scope of this paper to go deeper into architecture style.

The API layer is responsible for extracting data from the repositories, analyzing, and calculating metrics and statistics over the ecosystem. This layer provides



(a) Souce codes information.



(b) Navigation.

Fig. 6. Source codes side by side view.

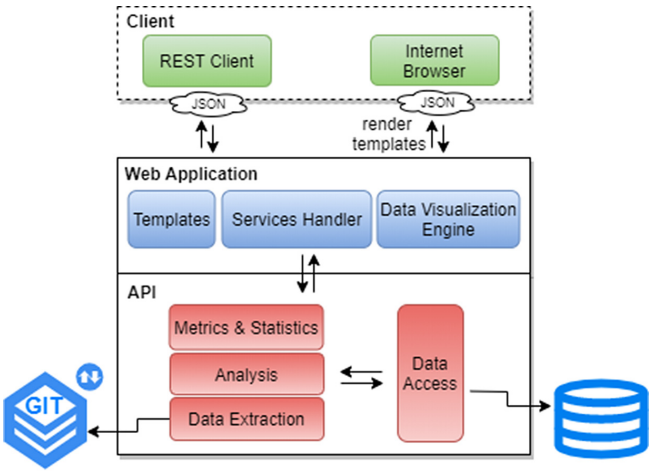


Fig. 7. *Ecolyzer* architecture.

database services for data access and storage. Currently, the API layer supports only Git[1] repositories and the Lua[2] programming language.

The web application tier (or the back-end) is a web service for data handling and visualization. It is responsible for handling requests and responses from the clients and interacting with the API layer. The provided services can be accessed either by RESTful services or by web templates.

Finally, the client tier can be either an internet browser or a specific application that consumes REST services. This layer provides the visualization to the end-users by the web templates, which are HTML files that contain static data and placeholders for dynamic data rendering on the browser [10]. The graphs are provided by the data visualization engine D3.js [27].

4 Evaluating API Usage for TerraME

To demonstrate the feasibility of our approach, we conducted an exploration of the relationships of a SECO. The main purpose of this Proof of Concept (PoC) was to highlight the usage of the proposed visualizations in a real case project. We expected that the visualizations would aid in analyzing the API usage considering its ecosystem.

4.1 TerraME

TerraME (Terra Modeling Environment) is an open-source platform API to develop and simulate spatially explicit dynamic models based on cellular automata, agents, and network models [7]. TerraME provides a Lua code interpreter [13], as well as functions that ease the development of multi-scale and multi-paradigm models for environmental applications. Contributions to TerraME can be encapsulated in packages and might extend its functionalities or provide new resources and concepts through its API.

TerraME is part of the National Institute for Space Research (INPE) software ecosystem, which develops and provides a set of systems in the geoprocessing area. In addition to being part of an ecosystem, TerraME has its own ecosystem, specifically focused on environmental modeling.

All software of the TerraME ecosystem are publicly available on GitHub, consisting of ten packages developed by third parties and two software that only uses its functions. Among the software, we highlight INPE-EM (INPE - Emission Model), that produces annual estimates of greenhouse gas (GHG) emissions due to changes in land cover in Brazil [2], and LuccME, an open-source framework for spatially explicit land use and cover change (LUCC) modeling [1].

[1] https://git-scm.com.
[2] https://www.lua.org.

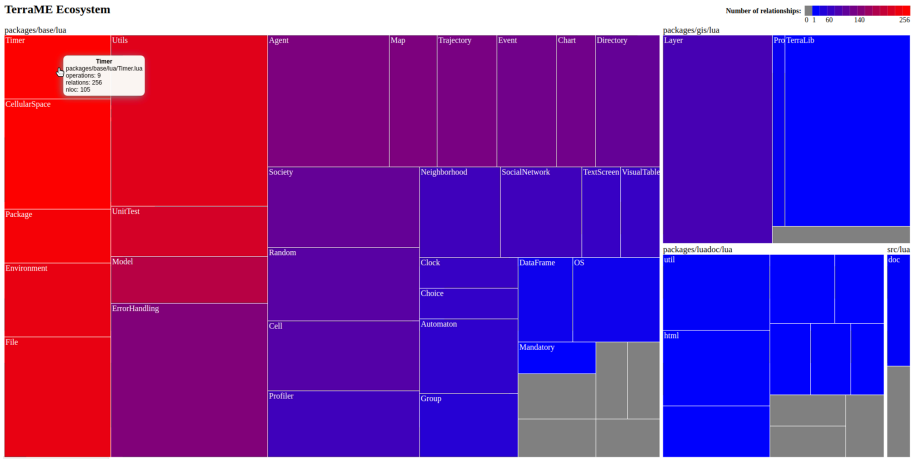


Fig. 8. TerraME heatmap. (Color figure online)

4.2 Proof of Concept

The extraction process starts by cloning the repositories and setting TerraME as the central system. We cloned the master branches of all software on January 5, 2020. Then, Ecolyzer scans the repositories and creates all relationships. After that, the ecosystem is ready for visualization on an internet browser. Figure 8 shows the heatmap of TerraME components.

Unlike object-oriented languages, where functions (or methods) are usually defined within classes, TerraME source files can be seen as components in which functions can be defined without the use of a structure like a class. By convention, TerraME defines most of its functions within Lua tables, which work similarly to classes.

On the API heatmap view (Fig. 8), it is possible to observe that TerraME has four modules composed of 52 components, 520 public functions, and 3658 relationships with other systems. The minimum and the maximum number of relationships among components is one and 256, respectively, as shown in the legend on upper right. Besides, ten components have no relationship at all (gray), eight components have a large number of connections (red), and the remainder has a low or medium number of relationships (blue).

Also as a result of the experiment: (1) the component with the most relationships is Timer, used by 180 components from nine software; (2) the system with more components that uses Timer is LuccME, with 38 components and 181 relationships; (3) the system that uses less components is INPE-EM, with only one component and one function call; (4) the component that most uses the Timer functions is the MultipleRuns component of the calibration package, with 26 function calls; (5) most software use only one Timer function; (6) the LuccME software components use the most number of different Timer functions. Figure 9 shows the component with the most relationships and the software that uses it.

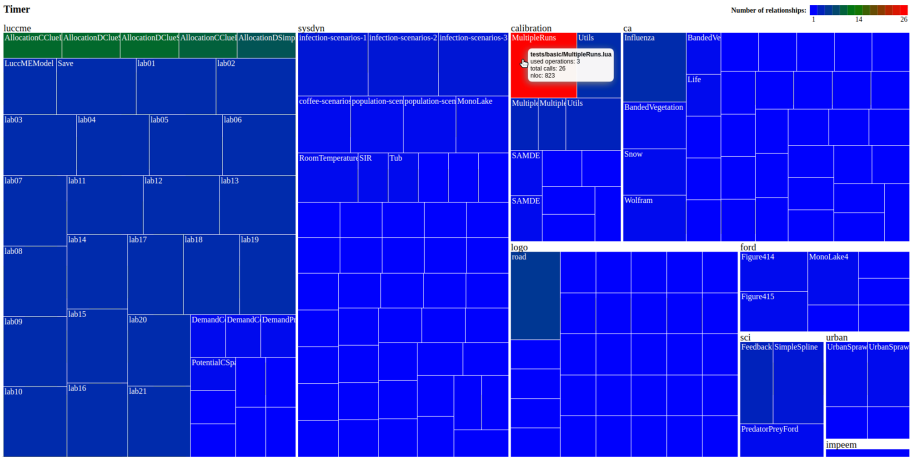


Fig. 9. Software with relationships on Timer component.

Our exploratory analysis through visualization has successfully provided important data about TerraME's usage, considering its ecosystem. Besides, we conducted an analysis together with the TerraME team in order to find out if our tool was able to provide more interesting feedback for the team involved in its development.

4.3 Insights for TerraME Ecosystem

To test the hypothesis that one of the main benefits provided by SECO visualization is to provide systematic insights, we introduced Ecolyzer to the TerraME development team and requested feedback on our tool. No protocol was followed, as we believe it would facilitate insights. We presented only some concepts of SECO, the tool, its functionalities, and how it works.

The team's first feedback reported that the TerraLib component, which is a facade for an API to access geographic data developed for the exclusive use of internal components, was strangely being used by two external packages. According to the development team, the functions of the facade is exposed through more specialized components in the most user-friendly way possible for end-users or other software that use TerraME, and should not be accessed directly by other software.

The TerraME's development team delivered other reports, such as: (1) *"the components supposed to be used a lot are, in fact, extensively used by other software"*; (2) *"some components in which we spent much work on them are not being used"*; (3) *"several new components with interesting functionalities have low usage, maybe we should publicize them better"*; (4) *"we could use this tool to measure the impact of changing any interface, which was not feasible so far"*; (5) *"we can use this tool to see how the functions are being used by other software"*;

(6) *"this tool might be helpful for new developers to understand how functions are used in practice"*.

This feedback allowed us to identify an important feature for those who analyze an API considering its ecosystem. It consisted of a filter on the functions provided by the components of the API. With this functionality, it would be possible to select a function provided by the component, and the heatmap would display only the software that uses this function, thus facilitating the measurement of the impact of its changes.

5 Conclusion and Future Work

There are several studies related to software ecosystems, and they can help the analysis and understanding of complex relationships among system dependencies. Furthermore, they come as an alternative to overcome challenges during the design, development, maintenance, and evolution of software, mainly on the ones that provide functions and services. However, there is still a lack of software tools to support software ecosystems analysis and visualization, thus making software engineering a critical research issue in the SECO domain.

This paper presented Ecolyzer, a software visualization tool for the visualization and analysis of API usage considering an ecosystem. A brief description of its architecture was presented, which was designed and implemented based on the directives proposed in the literature on SECO visualization tools.

The visualization provided by Ecolyzer was reported in detail. A proof of concept (PoC) in a real open-source platform API was conducted, demonstrating that the tool is useful for evaluating an API usage in the context of a SECO, allowing to quantify by who and how the API is actually used. Also, Ecolyzer proved to be useful for measuring the impacts of changes in API functions, providing insights on its usage, thus helping the organization that maintains the API to plan its evolution strategically.

As future work, we intend to (1) add a new feature to the tool that enables it to analyze the historical evolution of an API and its ecosystem using the commits metadata, (2) implement support for other programming languages to carry out studies in other ecosystems, and (3) investigate new metrics of API usage in the context of SECO. Finally, we plan to conduct a systematic usability evaluation of Ecolyzer.

Acknowledgment. The authors would like to thank the World Bank (grant #P143185) and FAPESP (grant 2018/22064-4) for financial support.

References

1. Aguiar, A., Carneiro, T., Andrade, P., Assis, T.: Luccme-terrame: an opensource framework for spatially explicit land use change modelling. GLP News **8**, 21–23 (2012)

2. Aguiar, A., et al.: Modeling the spatial and temporal heterogeneity of deforestation-driven carbon emissions: the INPE-EM framework applied to the Brazilian Amazon. Global Change Biol. **18** (2012). https://doi.org/10.1111/j.1365-2486.2012.02782.x
3. Barbosa, O., Alves, C.: A systematic mapping study on software ecosystems. In: Proceedings of International Workshop on Software Ecosystems (2011)
4. Berger, T., et al.: Variability mechanisms in software ecosystems. Inf. Softw. Technol. **56** (2014). https://doi.org/10.1016/j.infsof.2014.05.005
5. Bloch, J.: How to design a good API and why it matters. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 2006, pp. 506–507. Association for Computing Machinery, New York (2006). https://doi.org/10.1145/1176617.1176622
6. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference, SPLC 2009, pp. 111–119. Carnegie Mellon University, USA (2009)
7. Carneiro, T.G.d.S., de Andrade, P.R., Câmara, G., Monteiro, A.M.V., Pereira, R.R.P.: An extensible toolbox for modeling nature-society interactions. Environ. Model. Softw. **46**, 104–117 (2013). https://doi.org/10.1016/j.envsoft.2013.03.002
8. García-Holgado, A., García-Peñalvo, F.J.: Mapping the systematic literature studies about software ecosystems. In: Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality, TEEM 2018, pp. 910–918. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3284179.3284330
9. Goeminne, M., Mens, T.: A framework for analysing and visualising open source software ecosystems. In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), IWPSE-EVOL 2010, pp. 42–47. Association for Computing Machinery, New York (2010). https://doi.org/10.1145/1862372.1862384
10. Grinberg, M.: Flask Web Development: Developing Web Applications with Python. O'Reilly Media, Inc. (2018)
11. Hammouda, I., Knauss, E., Costantini, L.: Continuous API design for software ecosystems. In: 2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering, pp. 30–33, May 2015. https://doi.org/10.1109/RCoSE.2015.13
12. Hanssen, G.K.: A longitudinal case study of an emerging software ecosystem: implications for practice and theory. J. Syst. Softw. **85**(7), 1455–1466 (2012). https://doi.org/10.1016/j.jss.2011.04.020. Software Ecosystems
13. Ierusalimschy, R., de Figueiredo, L.H., Filho, W.C.: Lua - an extensible extension language. Softw. Pract. Exper. **26**(6), 635–652 (1996). https://doi.org/10.1002/(SICI)1097-024X(199606)26:6⟨635::AID-SPE26⟩3.0.CO;2-P
14. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: a research agenda for software ecosystems. In: 2009 31st International Conference on Software Engineering - Companion Volume, pp. 187–190, May 2009. https://doi.org/10.1109/ICSE-COMPANION.2009.5070978
15. Jansen, S., Handoyo, E., Alves, C.: Scientists' needs in modelling software ecosystems. In: Proceedings of the 2015 European Conference on Software Architecture Workshops. ECSAW 2015. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2797433.2797479
16. Knodel, J., Manikas, K.: Towards a typification of software ecosystems. In: Fernandes, J.M., Machado, R.J., Wnuk, K. (eds.) ICSOB 2015. LNBIP, vol. 210, pp. 60–65. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19593-3_5

17. Lungu, M.: Towards reverse engineering software ecosystems. In: 2008 IEEE International Conference on Software Maintenance, pp. 428–431, September 2008. https://doi.org/10.1109/ICSM.2008.4658096
18. Lungu, M., Lanza, M., Gîrba, T., Robbes, R.: The small project observatory: visualizing software ecosystems. Sci. Comput. Program. **75**(4), 264–275 (2010). https://doi.org/10.1016/j.scico.2009.09.004
19. Manikas, K.: Revisiting software ecosystems research: a longitudinal literature study. J. Syst. Softw. **117**, 84–103 (2016). https://doi.org/10.1016/j.jss.2016.02.003
20. Messerschmitt, D., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, Cambridge (2003). https://doi.org/10.7551/mitpress/6323.001.0001
21. Oscar, F.B., Ameller, D., Costal, D., Franch, X.: Open source software ecosystems: a systematic mapping. Inf. Softw. Technol. (2017). https://doi.org/10.1016/j.infsof.2017.07.007
22. Pérez, J., Deshayes, R., Goeminne, M., Mens, T.: Seconda: software ecosystem analysis dashboard. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 527–530. IEEE, March 2012. https://doi.org/10.1109/CSMR.2012.69
23. Richardson, L., Amundsen, M., Amundsen, M., Ruby, S.: RESTful Web APIs: Services for a Changing World. O'Reilly Media, Inc. (2013)
24. da Silva Amorim, S., McGregor, J.D., de Almeida, E.S., von Flach, G.C.C.: Software ecosystems' architectural health: another view. In: 2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS), pp. 66–69, May 2017. https://doi.org/10.1109/JSOS.2017.15
25. Santana, F.W., Werner, C.M.: Towards the analysis of software projects dependencies: an exploratory visual study of software ecosystems. In: International Workshop on Software Ecosystems (IWSECO), vol. 987, pp. 7–18. Citeseer (2013)
26. Spadini, D., Aniche, M., Bacchelli, A.: PyDriller: python framework for mining software repositories. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018, pp. 908–911. ACM Press, New York (2018). https://doi.org/10.1145/3236024.3264598
27. Zhu, N.Q.: Data Visualization with D3.js Cookbook. Packt Publishing Ltd (2013)