Assessing Open Source Communities' Health using Service Oriented Computing Concepts

Marc Oriol, Oscar Franco-Bedoya, Xavier Franch and Jordi Marco
Universtat Politècnica de Catalunya
Barcelona, Spain
{moriol, ohernan, franch}@essi.upc.edu, jmarco@lsi.upc.edu

***Abstract*—The quality of Open Source Software products is directly related to its community's health. To date, health analysis is made accessing available data repositories or using software management tools that are often too static or ad hoc. To address this issue, we propose to adopt principles and methods from the Service Oriented Computing field. Particularly, we propose to adapt the concepts of quality service and service level agreement, and propose to reuse the existing body of knowledge and techniques from SOC monitoring. To demonstrate the feasibility of the approach, we use a service monitoring framework called SALMonOSS as a proof of concept to realize the implementation of the proposal.**

Keywords—Monitoring, Open Source Software, Community Health, OSS, SOC

I. Introduction

Open Source Software (OSS) has become a strategic asset for a number of reasons, such as its short time-to-market software service and product delivery, reduced development and maintenance costs, introduction of innovative features and its customization capabilities. By 2015, an estimated 85% of all commercial software packages will include OSS components [1]. Under these circumstances, ensuring OSS quality becomes crucial.

Quality assessment is always a difficult endeavour, and it is even more difficult in the OSS arena, where new patches and releases are constantly deployed and an increasing number of niche players want to participate in the resulting software community [2]. An aspect that makes OSS quality analysis different is the existence of communities. OSS components are managed and distributed in a collaborative way by an OSS community following some particular communicational processes and involving some typical resources. Contributors communicate bugs, committers confirm patches, the community produce releases, branches/forks may occur at any moment, etc. Community members interact using mailing lists, blogs, forums, etc. These processes and resources need to be considered in a comprehensive quality assessment program. It has been already argued that quality and sustainability of OSS components heavily depends on the behaviour and evolution of OSS communities along time [3], i.e., their health. Being able to effectively assess the health of OSS communities is the ultimate objective of our work.

Health is a term from biology, which refers to the status of an ecosystem and is a metaphor in the context of OSS communities [4]. OSS ecosystem health operationalization, using the data from collections of open source projects that belong to the ecosystem, is provided by Jansen in [5]. Perez et al. studied the correlation between OSS community health and software quality [6]. Abreu and Premraj showed a statistically significant correlation between communication frequency and number of injected bugs in the software [7]. This example illustrates that software development is a social-technical activity, where there exists a statistically significant correlation between social activities and software quality [8]. It also shows how OSS community health diagnostic requires monitoring the data related to the community such as mailing lists, bug tracking systems and version repositories [9], for this reason it is necessary to develop tools that collect the values of OSS community metrics anytime.

There has been in the last decade a considerable amount of research, especially of quantitative nature, instrumenting OSS community health analysis through a set of metrics (e.g., average time to resolve a critical bug), which are used in real scenarios to improve the quality of the OSS [10]. Still, this existing work mainly focuses on engineering accurate and efficient solutions to the data gathering problem, e.g. how to access to resources like bug trackers, how to analyse community members' messages in order to find relevant information (e.g., sensitivity analysis), etc. Being this necessary, there is also a need of well-established solutions to:

1) Manage this set of metrics along time according to some pre-established plans.
2) Link the gathered values with client's needs.
3) Engineer a portfolio of methods and techniques to support OSS communities and OSS adopters in the exploitation of this data.
4) Avoid very large list of metrics by defining a set of key health indicators to evaluate the OSS communities' health.

To solve these issues, we envision that the knowledge, practices and methods in other fields with similar challenges can provide a robust solution to accomplish the aforementioned goals. Particularly, we believe that the current state of the art in Service Oriented Computing (SOC) related to quality assessment has strong similarities that can be ported into OSS community health analysis.

To this aim, in this paper we present a new framework, named SALMonOSS, which adopts the techniques from monitoring Quality of Service (QoS) of services to monitor the OSS community health. By following analogue principles, we argue that a set of techniques and applications based on top of the monitored data can be also ported from SOC to OSS.

The paper is organized as follows: In section II we present the related work of quality assessment for both OSS and SOC. In section III, we discuss the alignment we envision in the two fields. We present the proposed framework, SALMonOSS, in section IV, including some preliminar results. In section V we describe a list of applications that can benefit from our approach. Finally conclusions and future work are discussed in section VI.

II. Related work

This section provides an overview of the related work in the fields of OSS and SOC.

A. Assessment of OSS community health

The assessment of OSS community health is usually realized by tools for a particular community or for a specific platform. For instance, there exists several solutions in the literature for monitoring and analysis of specific OSS communities by accessing directly to their available data repositories, e.g. Nagios [3], GNOME [8], Ruby [11], Debian [12], Eclipse [13] and Sourceforge [14]. Generally, these tools are specifically designed for one or a few scientific experiments [8]. Consequently, these tools are not available once the research is finished and are not suitable to monitor other communities.

On the other hand, there are frameworks and tools that automate the monitoring and analysis over specific repositories, e.g. FLOSSMETRICS (flossmetrics.org), QUALIPSO (qualipso.org), QUALOSS (qualoss.eu), the ongoing OSSMETER (ossmeter.eu), LTC (passion-lab.org/projects), MARKOS (markosproject.eu), Ohloh (ohloh.net) and finally Goeminne and Mens [8]. Each work introduces a generic and extensible framework for studying OSS communities. Although these tools are more generic and reusable, most of the presented works are limited to specific technologies (e.g. Bugzilla, CVS, Git, Jira) that are monitored separately. Hence, the analysis is performed without a clear picture of the system as a whole (e.g. an OSS community may decrease the number of forum posts because they tend to use more the mailing list and not because the community shrinks).

B. Quality assessment in SOC

In the field of SOC, there are several works addressing the challenges of monitoring the QoS of services and the analysis of the compliance of their Service Level Agreements (SLAs) (which is a document that states the levels of acceptance with respect to the QoS). The automation of monitoring has been already achieved by different works (either by gathering the data using online testing [15], passive monitoring [16] or a combination of both). Most of the monitoring solutions are domain-independent, and can be used for different services regardless the domain. The analysis of the SLAs has also been addressed in depth, eventually providing a comprehensive explanation of the cause of SLA violations [17].

To the best of our knowledge, there is only one work, from Ghezzi et al. [18], that applies the idea of implementing data repositories of an OSS into services. In this work, they present a framework named SOFAs, that implements the different repositories as RESTful services to monitor the evolution of OSS. However, their work is restricted to software evolution, and the analysis performed is not suitable to check the OSS community health.

III. Aligning OSS and SOC

A. OSS Health

Key Performance Indicators (KPIs) can determine the health of an ecosystem. For example, Iansiti and Levien have introduced three determinants KPIs of business ecosystem health: robustness, productivity and niche creation [19]. The concept Key X Indicator is not only related to performance. Ferguson et al. used the KPI for quantifying uncertainty in early lifecycle cost estimation [20], Raanan and Kenett used Key Risk Indicators (KRIs) for monitoring risks in business unit [21]. Furthermore, KRIs were used in the RISCOSS Analytics tool for the development of a risk management approach applicable to the adoption and deployment of OSS [22]. In this work, we have defined the concept Key Health Indicators (KHIs) as a valid instrument to evaluate the OSS community health. This is inspired by a Software Engineering Institute (SEI) approach called QUELCE [20] for handling early stage project planning estimation. Consequently, KHIs may support strategic decision making within the OSS community and the OSS adopter. To this aim, KHIs should be continuously monitored to evaluate the OSS community health.

The list of health indicators may be very large. Therefore, the KHIs should be reduced to an efficient set of indicators that capture the impact on the OSS community health. To do this, in first place, values of OSS community variables are extracted from data and metadata saved in OSS community repositories (e.g., version control systems, mailing lists, bug trackers, websites, wikis, discussion forums). Secondly, social network data is generated using social network analysis tools (e.g., NodeXl [23]). Subsequently, KHIs are calculated using Bayesian networks by fusion of community variables and domain expert knowledge to support the definition of relations between OSS community measures and KHIs [20] (e.g., number of commits and activeness of the community).

We have defined an architecture for extracting and processing OSS data, based on the 3-layered approaches proposed by Franch et al. [24] and Geommine and Mens [8] (See Fig. 1). Layer I is for extracting raw data from OSS community repositories. In layer II, the measurements are processed and structured, e.g. in terms of the member relationships collected from community social media. This social data is gathered via natural language processing techniques. In layer III, the KHIs are calculated using Bayesian networks and OSS experts assessment based on their experience in OSS communities.

B. Alignment with SOC

Fig. 2 shows graphically how we substantiate the idea of aligning SOC and OSS. In the field of SOC, illustrated in the left-hand side of the figure, services are provided by service providers, who must guarantee a certain quality of the service (QoS) as to fulfill the client's needs about performance, availability, etc., which are operationalized into an SLA. An SLA is composed of a set of Service Level Objectives (SLOs), which constrain the permissible values that service metrics (throughput, response time, etc.) may take. The QoS is computed in a regular basis using a monitor, which

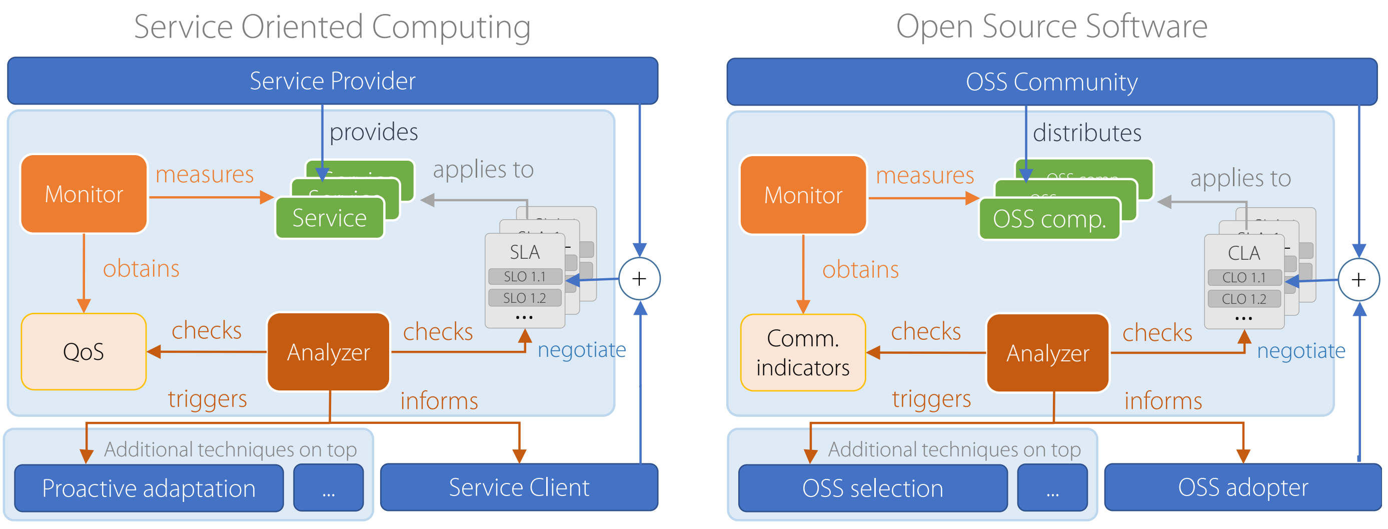

Fig. 2. Service and OSS monitoring

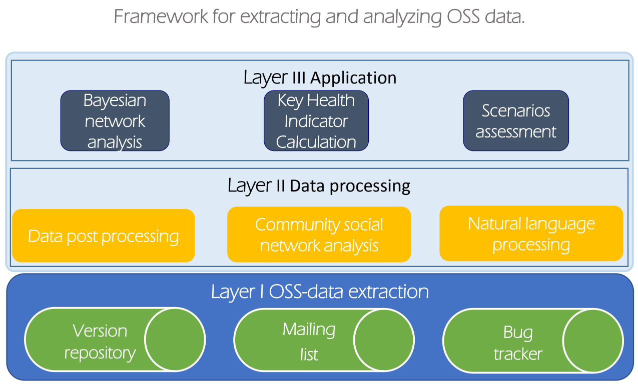
Framework for extracting and analyzing OSS data.
Layer III Application
Bayesian network analysis
Key Health Indicator Calculation
Scenarios assessment
Layer II Data processing
Data post processing
Community social network analysis
Natural language processing
Layer I OSS-data extraction
Version repository
Mailing list
Bug tracker

Fig. 1. Architecture for extracting and analyzing OSS data.

measures the behaviour of the services during their execution. The monitoring results are checked against the SLA by an analyser, and possible violations are detected and reported. On top of this basic schema, several tasks as service selection [25], and techniques as proactive adaptation [26], may benefit of the monitoring infrastructure. In the right-hand side of the figure, we may see that OSS communities are the counterpart of service providers, since they distribute the OSS component outside. OSS adopters (the counterpart of service clients) need to operationalize the quality that they demand to the OSS component in terms of what we may call Component Level Objectives (CLO) which together conform a Component Level Agreement (CLA).

As said above, in this paper we focus on those CLOs that involve metrics that are bound to the information gathered from the community through resources like bug trackers and forums (mean time to repair a bug, volume of messages per day,...). These metrics can be aggregated to evaluate and assess KHIs (like timeliness and activeness). On top of this scenario, several tasks as OSS selection, and techniques as social network analysis, may benefit from this monitoring schema.

IV. SALMonOSS

To prove the feasibility of the approach, we haved implemented a framework named SALMonOSS. SALMonOSS is an OSS Health monitoring framework based on an existing technology named SALMon [27] developed in our research group. SALMon is a versatile monitoring framework to monitor the QoS of services. Here we describe how we have aligned the problem of assessing OSS community health with service monitoring, and the enhancements of SALMon to support OSS monitoring, resulting in the SALMonOSS framework.

A. General requirements

The process of monitoring the OSS community health has a set of high-level requirements that must be addressed to properly develop a monitoring solution for OSS. Based on the different aspects of the monitoring process in SOC and their applicability to the OSS domain, we have identified the set of requirements that are briefly described bellow.

Servizitation. OSS development and maintenance usually requires different software management tools, such as mailing lists, bug tracking systems and version repositories; each one providing an aspect of the OSS community health. **Requirement 1.** OSS management tool functionalities shall be offered as services. Options are: 1) the tool already provides a web service interface (e.g., the JIRA bug tracking system provides a RESTful service), 2) the management tool is wrapped into a web service (see Fig. 3). By monitoring services instead of components, OSS community health can be assessed by current service monitoring technologies with minimal changes.

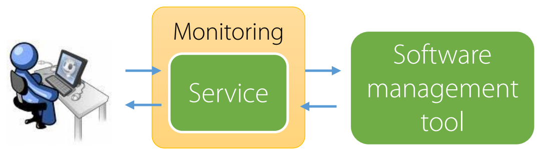

Fig. 3. Wrapping a tool into a service

Strategies. SOC provides two strategies to conduct monitoring, active monitoring and passive monitoring. **Requirement 2.** In order to not losing the potential of SOC monitoring, SOC strategies need to be applicable in OSS community health analysis. The meaning needs to be slightly adapted, though:

- Active: Under this approach, monitoring can be performed without the involvement of the OSS community. A software component invokes periodically a set of operations of the services that retrieve the status of the OSS. By monitoring the results of these invocations, the metrics related to the OSS community health are computed. This is analogous to active monitoring (or online testing) in SOC.
- Passive: Under this approach, OSS communities are required to manage the development and maintenance of the OSS using the wrapping services (e.g. any new bug is reported using the service interface). Transparently to the service client, services are continuously monitored and whenever a new event occurs, the metrics related to the OSS community health are calculated. This is analogous to passive monitoring in SOC.

It is worth to remark that these approaches are not mutually exclusive and they can be combined. That is, when monitoring the health of an OSS community, some data can be retrieved using the passive approach and the rest of the data using the active approach.

Extensibility. It is difficult to predict in advance the full set of metrics that may need to be monitored. We can expect that this may be even more difficult in the future since the field of OSS community health analysis still has room for improvement. **Requirement 3.** OSS community health analysis tools need to be extensible to host new metrics.

Interoperability. In SOC, monitoring tools are the basis for applying more sophisticated treatments: self-adaptation, prediction, etc. This potential should not be lost when adapting to the OSS field. **Requirement 4.** OSS community health analysis tools need to support interoperability with other, potentially unknown, tools. This way, we can think on supporting tasks like assessing in the decision of going or not for an update in a given moment, or to decide to start risk mitigation actions when some community indicator goes beyond some threshold.

CLA-aware. We have already explained the importance that SLA has in SOC monitoring. SLAs are a way to express the expectations of end users on the monitored system, and this concept seems also useful for our OSS context. **Requirement 5.** OSS community health analysis tools need to be customized in a particular deployment according to the contents of the CLA. Therefore, the tool will just monitor the metrics that are required to check the CLA, that is the ultimate goal of the monitoring process.

B. Monitoring process

SALMonOSS supports both passive and active approaches to monitor the required services(Req 2). To combine both approaches (see Fig. 4), the *Invoker* (1a) uses the same infrastructure as used by the *OSS community* (1b). In both cases, the invocations are performed through the same *Enterprise Service Bus (ESB)*, which is a middleware able to handle and manage the communication between services. When receiving a service request, the *ESB* notifies to the *Monitor* the event (2). The *Monitor* then, notifies it to the *Measure Instruments* that are responsible to handle such request (3). In parallel to the previous monitoring activities, the *ESB* forwards the service request to the *Software management tool*, who may have been wrapped if it is not providing a service itself (Req 1) (4). The *service* invokes the software management tool (5), and receives the response (6). The response is sent through the *ESB* (7) to the initiator of the request, either the *Invoker* or the *OSS community* (8). In parallel, the same response is notified to the *Monitor* (9), which notifies it to the *Measure Instruments* (10), which ultimately calculates the measurements (11).

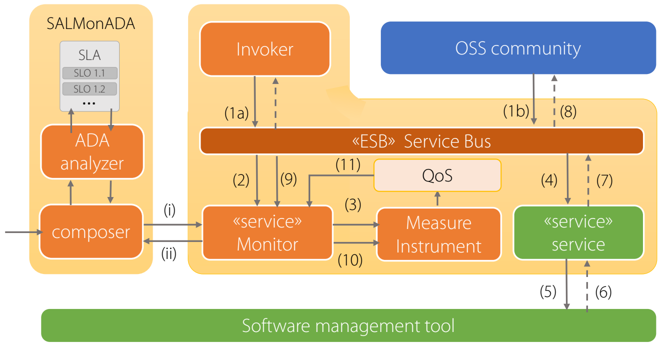

Fig. 4. OSS monitoring process in SALMonOSS

SALMon has been implemented as a service based system by itself, and so does SALMonOSS. By following the SOA principles, SALMonOSS can be easily integrated to different frameworks that require monitoring (Req 4), as happened with SALMon that was used to support self-adaptive service based systems [28], federated cloud management [29], or service selection [25], among others. On the other hand, being the monitor service an aggregate of measure instruments, satisfies Req 3: measure instruments are pluggable components that deal with a particular metric each, so that the monitor instantiates and manages the required set of measure instruments. By following this approach, new metrics can be calculated by implementing the required measure instruments.

Finally, SALMonOSS can be automatically configured to monitor the metrics included in a CLA, with the combination of the ADA software (Req 5). ADA is an Agreement Document Analysis framework aimed at extracting useful information from agreement documents [17], and with SALMon constitute a combined framework named SALMonADA. Under this framework, the CLA is parsed by the Composer, and the Monitor is configured through its interface (i). Whenever new measurements are computed, they are reported to ADA through the composer (ii) in order to perform the analysis of the CLA fulfillment.

For example, with SALMonOSS it is possible to monitor metrics of the OSS community such as *commits per day* or *time to resolve a bug*. When a new commit or bug is captured by the ESB (either by passive or active approach), the Measure Instruments compute the metrics, which are then analysed against the CLA to check if there is any violation over the conditions which would compromise the KHIs. In such a case, the interested parties are notified.

C. *Implementation and preliminary results*

The current implementation of the prototype is focused on monitoring mailing lists, bug tracking systems and version repositories. Particularly, the technologies chosen are Markmail, Jira and GIT, respectively. All these technologies are required to be used as services. Jira already provides a RESTful service interface, whereas Markmail and GIT tools have to be wrapped. With respect to the GIT tool, we have implemented a service that wraps the GIT with the methods to retrieve the details related to commits. Preliminary results of monitoring the GIT repository using the active monitoring strategy are shown in Fig. 5. We have monitored the behaviour of the commits performed, which is an indicator for the activeness of the community, on the xwiki[1] during the month of December. Particularly we have monitored the following metrics: number of commits per day, files changed per day, lines added per day, lines removed per day, average files changed per commit, average lines added per commit and average lines removed per commit. As shown, we can observe that the community is active during weekdays, but inactive during weekends and holidays.

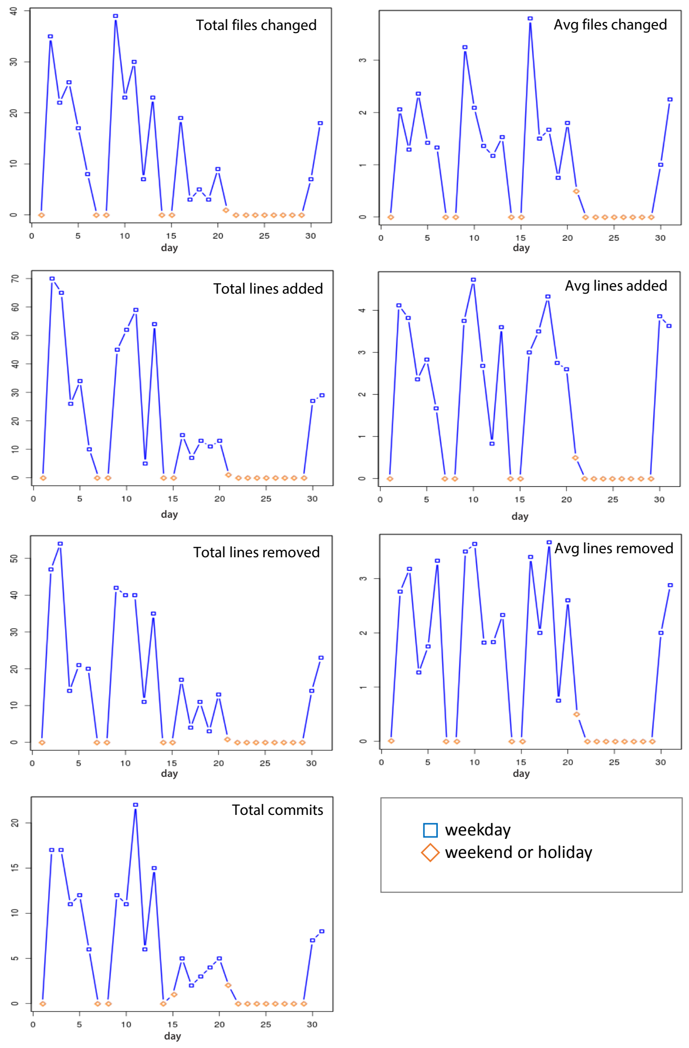

Fig. 5. Metrics related to GIT monitored with SALMonOSS

All the retrieved information is then checked against the CLA, in order to identify possible deviations and eventually assess their KHIs.

V. Applications

Applying the same paradigm from SOC to OSS puts forward the capability to transfer and reuse the knowledge and techniques from one field to the other. Particularly interesting are those techniques which have a straightforward resemblance on their objectives. We describe below some clear applications:

A. *OSS selection*

As a first step for OSS adoption, a particular OSS has to be selected. There are different OSS that fulfill the same functional requirements for an OSS adopter, hence the selection of an OSS should be based not only on the functional requirements but also on the quality and sustainability of the OSS, which is directly related to the health of the OSS community. The selection process can be facilitated in an automated manner through technologies that implements algorithms based on multiple-criteria decision analysis. There are several frameworks that implements those algorithms in the field of service selection [25], which we argue could be applied in the field of OSS.

B. *OSS violation prediction*

During the execution of the OSS, violations of the CLA can be forecasted before they occur by applying analysis over statistical models using the monitored data. The results of such forecasting can be reported to the OSS community, who in turn, can apply the required mechanisms to mitigate any risk and avoid such violations (e.g. reallocating resources and assigning new priorities). There exists different techniques in the field of SOC that predict violations of SLAs [30], we believe that the same techniques could by applied to predict violations of CLAs for OSS.

C. *OSS adaptation*

In case of a CLA violation, the OSS adopter might be interested on performing an adaptation to reestablish the indicators specified in the CLA. For instance, if there's a critical security bug on an OSS component without a clear commitment by the OSS community to solve it in a short period of time, the OSS adopter might replace or disable the OSS until the issue is solved. In the field of SOC, automatic adaptation of services has been a major topic of study. In SOC, when a SLA is violated, an adaptation need is triggered, which prompts the generation of an adaptation plan that is ultimately enacted by a component able to execute the adaptation [31]. Although service and OSS adaptation have their differences in terms of technology and standardization, the progress and results made could be reused and applied in the field of OSS.

[1]xwiki is a free wiki software platform developed and maintained with the support of an OSS community (http://www.xwiki.org/)

VI. Conclusions and Future Work

In this paper we demonstrate how current SOC monitoring techniques can be ported and used to assess the OSS community health. Particularly, we have presented SALMonOSS, an OSS community health monitoring framework able to: (1) monitor a list of community health metrics along time (2) link the gathered values with client's needs by operationalizing conditions in CLAs and (3) engineer a portfolio of methods and techniques that supports OSS communities and OSS adopters (e.g. OSS selection, proactive adaptation, etc.). Furthermore, the solution follows a layered architecture that structures the data from low-level metrics to KHIs.

As future work we plan to extend the list of services to monitor in order to include more data repositories, implement bayesian network analysis for KHIs, and apply some of the techniques described that benefits from the monitoring data (e.g. OSS selection).

Acknowledgments

This work is a result of the RISCOSS project, funded by the EC 7th Framework Programme FP7/2007-2013, agreement number 318249. Oscar Franco-Bedoya has a grant from Universidad de Caldas, Colombia. We would also like to thank the student Francesc Trull for his contribution on the implementation of the prototype.

References

[1] M. Driver, "Hype cycle for open-source software," Gartner, Tech. Rep., 2013.

[2] S. Jansen and G. van Capelleveen, "Quality review and approval methods for extensions in software ecosystems," in Software Ecosystems Analyzing and Managing Business Networks in the Software Industry. Edward Edgar Publ. Lim., 2013, p. 187.

[3] J. Gamalielsson, B. Lundell, and B. Lings, "The Nagios community: An extended quantitative analysis," in Open Source Software: New Horizons. Springer, 2010, pp. 85–96.

[4] D. Wynn Jr., "Assessing the Health of an Open Source Ecosystem," in Emerging Free and Open Source Software Practices, S. K. Sowe, I. G. Stamelos, and I. Samoladas, Eds. IGI Global, Jun. 2007, pp. 238–258.

[5] S. Jansen, "Measuring the health of open source software ecosystems: Moving beyond the project scope," Information and Software Technology, p. in Press, 2014.

[6] J. Pérez, R. Deshayes, M. Goeminne, and T. Mens, "Seconda: Software ecosystem analysis dashboard," in Software Maintenance and Reengineering (CSMR). Szeged,Hungary: IEEE, 2012, pp. 527–530.

[7] R. Abreu and R. Premraj, "How developer communication frequency relates to bug introducing changes," in Proceedings of the joint international ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM, 2009, pp. 153–158.

[8] M. Goeminne and T. Mens, "A framework for analysing and visualising open source software ecosystems," in 13th International Workshop on Principles on Software Evolution. ACM, 2010, pp. 42–47.

[9] C. Jergensen, A. Sarma, and P. Wagstrom, "The onion patch: migration in open source ecosystems," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE). ACM, 2011, pp. 70–80.

[10] P. Rotella and S. Chulani, "Implementing quality metrics and goals at the corporate level," in Proceedings of the 8th Working Conference on Mining Software Repositories (MSR). ACM, 2011, pp. 113–122.

[11] J. Kabbedijk and S. Jansen, "Steering insight: An exploration of the ruby software ecosystem," in International Conference on Software Business (ICSOB), 2011, pp. 44–55.

[12] E. Ververs, R. van Bommel, and S. Jansen, "Influences on developer participation in the debian software ecosystem," in Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES). ACM, 2011, pp. 89–93.

[13] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg, "Shades of gray: Opening up a software producing organization with the open software enterprise model," Journal of Systems and Software, vol. 85, no. 7, pp. 1495–1510, 2012.

[14] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, and et al., "An empirical investigation into a large-scale java open source code repository," in Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2010, pp. 11:1–11:10.

[15] M. Palacios, J. Garcia-Fanjul, J. Tuya, and G. Spanoudakis, "Identifying test requirements by analyzing sla guarantee terms," in IEEE 19th International Conference on Web Services (ICWS), 2012, pp. 351–358.

[16] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, and R. Yahyapour, "Establishing and monitoring slas in complex service based systems," in IEEE International Conference on Web Services (ICWS), 2009, pp. 783–790.

[17] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, and M. Rodriguez, "Comprehensive explanation of sla violations at runtime," IEEE Transactions on Services Computing (TSE), vol. PP, no. 99, pp. 1–1, 2013.

[18] G. Ghezzi and H. Gall, "A framework for semi-automated software evolution analysis composition," Automated Software Engineering, vol. 20, no. 3, pp. 463–496, 2013.

[19] M. Iansiti and R. Levien, "Keystones and dominators: Framing the operational dynamics of business ecosystems," Harvard Business School, 2002.

[20] R. W. Ferguson, D. Goldenson, J. M. McCurley, R. W. Stoddard, D. Zubrow, and D. Anderson, "Quantifying uncertainty in early lifecycle cost estimation (quelce)," Carnegie Mellon University, Tech. Rep., 2011.

[21] Y. Raanan, R. S. Kenett, and R. Pike, Operational Risk Management: an overview. John Wiley & Sons, Ltd, 2010, pp. 19–38.

[22] B. Nili, B. Yehuda, R. Ben, and R. Kenett, "Intermediate proposal for risk management techniques," RISCOSS Project, Tech. Rep., 2013.

[23] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave, "Analyzing (social media) networks with nodexl," in Proceedings of the fourth international conference on Communities and technologies. ACM, 2009, pp. 255–264.

[24] X. Franch, R. Kenett, F. Mancinelli, A. Susi, D. Ameller, R. Ben-jacob, and A. Siena, "A layered approach to managing risks in open source sofware projects," in Int. Conference on Open Source Systems, 2014.

[25] O. Cabrera, M. Oriol, X. Franch, L. Lopez, J. Marco, O. Fragoso, and R. Santaolaya, "Wessqos: a configurable soa system for quality-aware web service selection," Universitat Politècnica de Catalunya, Tech. Rep., 2011.

[26] L. Baresi and S. Guinea, "Self-supervising bpel processes," IEEE Transactions on Software Engineering (TSE), vol. 37, no. 2, pp. 247–263, 2011.

[27] M. Oriol, J. Marco, X. Franch, and D. Ameller, "Monitoring adaptable soa-systems using salmon," in Workshop on Service Monitoring, Adaptation and Beyond (Mona+), 2008, pp. 19–28.

[28] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based online testing for proactive adaptation of service-based applications," in Computer Software and Applications Conference (COMPSAC), 2011, pp. 582–587.

[29] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, and et al., "Enhancing federated cloud management with an integrated service monitoring approach," Journal of Grid Computing, vol. 11, no. 4, pp. 699–720, 2013.

[30] A. Metzger, E. Schmieders, O. Sammodi, and K. Pohl, "Verification and testing at run-time for online quality prediction," in Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube), 2012, pp. 49–50.

[31] A. Bucchiarone, C. Cappiello, E. Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: Main issues and requirements," in Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, 2009, pp. 467–476.