Towards Profiling Runtime Architecture Code Contributors in Software Projects

Quentin Perez[1][a], Alexandre Le Borgne[2][b], Christelle Urtado[1][c] and Sylvain Vauttier[1][d]

[1]*EuroMov Digital Health in Motion, Univ. Montpellier, IMT Mines Ales, Ales, France*

[2]*Digital Services, Altran Technologies, Blagnac, France*

Keywords: Empirical Software Engineering, Code Ownership, Software Architecture.

Abstract: Empirical software engineering has leveraged open software repositories to profile and categorize project contributors. The objective of our work is to conduct a similar but original study, focused on architectural contributions, to evaluate the profiles of contributors playing this specific development role and their evolution over time. This paper presents an approach to study a first kind of architectural contributions: deployment descriptors that define runtime architectures of applications. A categorization model is proposed, reflecting the importance of contributions based on data mined from code repositories (contents, timestamps, authors, etc.). Then, it groups contributors in several categories (profiles) and studies their evolution in projects over time. A case study is conducted on a selected long-life, quality project. It shows that the specific architectural development responsibility we measure is chosen and sustained by experienced and committed contributors. As a proof of concept, these results are very promising and will lead to broader scale studies in order to classify projects based on their management policies regarding architectural contributors.

1 INTRODUCTION

Architecture is regarded as a major concern for software quality (Garlan and Shaw, 1993; Taylor et al., 2009). Architects define roadmaps, technologies, components and their connections. They are makers and keepers of major design decisions. According to Booch (1996), *"every project should have exactly one identifiable architect, although for larger projects, the principal architect should be backed up by an architecture team of modest size"*. Abrahamsson et al. (2010) advocate that the architectural activity on a project also depends on contextual factors such as project size, business model, team distribution, rate of changes, etc.

On the other hand, empirical studies, fueled by open software repositories such as GitLab or GitHub, have shown that there are two categories of developers: minor contributors and major contributors (Bird et al., 2011; Foucault et al., 2014, 2015b). The evolution and management of these two categories over time impact project health and software quality. To our best knowledge, no comparable empirical study focuses on contributions to architecture development.

This paper presents a first step towards profiling architecture code contributors, focused on a specific kind of contribution: the runtime architecture of applications. This architectural contribution is explicitly supported by frameworks such as Spring[1] thanks to architecture deployment descriptors.

The remainder of this paper is organized as follows. Section 2 details the research questions addressed in this paper along with our main hypotheses. Section 3 presents the proposed code contributor categorization model. Section 4 defines a concrete implementation of this model in our chosen experimental context. Section 5 analyzes the results obtained empirically on a case study. Section 6 discusses threats to validity. Section 7 presents related work while Section 8 concludes with future work.

2 RESEARCH QUESTIONS

Different kinds of contributions can be mined to categorize and measure architectural activities in projects. For instance, contributions to architectural design decisions may be mined from the semantic content of issue tickets or messages (Poncin et al., 2011).

---

[a] https://orcid.org/0000-0002-1534-4821

[b] https://orcid.org/0000-0002-1823-2091

[c] https://orcid.org/0000-0002-6711-8455

[d] https://orcid.org/0000-0002-5812-1230

[1]https://docs.spring.io/spring/docs/current/spring-frame work-reference/

Contributions to project architecture (*e.g.*, modules, dependencies) may be mined from Maven files (Teyton et al., 2012). In the same way, contributions to code architecture may be mined from source code (Teyton et al., 2014; Mockus and Herbsleb, 2002).

In this paper, we focus on yet another kind of architectural contribution: the definition of the runtime architecture of applications. Our choice is driven by the existence of frameworks dedicated to application architecture definition, as the Spring framework in the Java ecosystem (Gupta and Govil, 2010; Le Borgne et al., 2018; Perez et al., 2019), commonly used in software industry. This framework provides tools (such as a set of Java annotations) and languages (such as a deployment descriptor XML dialect) that support the definition of the architecture instantiated to execute an application. These runtime architecture definitions are explicit and contributions to them are thus easy to measure.

RQ₁: Can contributor profiles be mined from contributions to runtime application architecture definitions?

So, our first hypothesis is that building metrics from these specific core architectural contributions may be significant enough to characterize contributors, and in turn projects, regarding architecture code development.

RQ₂: Is there a link between runtime architecture and code contribution importance?

Our second hypothesis is that in quality projects, architectural contributions should mainly be developed and maintained by dedicated and skilled members. *RQ₂* aims at verifying that this hypothesis is true for runtime application architecture definitions.

RQ₃: What is the turnover within the different contributor categories?

Turnover is a crucial issue for software development quality. Foucault et al. 2014, 2015b proved that external turnover has a negative impact on open source software projects whereas internal turnover has a positive impact on team development and quality. We study turnover in the different contributor categories to qualitatively examine their evolution.

3 CONTRIBUTOR CATEGORIZATION MODEL

3.1 Software Contribution Model

Changes between a project version and its predecessors (whether immediate or indirect) can be described by a set of **contributions** which link each modified line of code in each file to its author. Each contribution also identifies the nature, the content, the date and time of the change. Formally, a contribution c is defined as sextuplet $c(f, l, n, e, a, t)$ where:

- f is a file,
- l is the modified line of code in f,
- n is the line number of l in file f,
- e is the nature of the change on l: addition (ADD), deletion (DEL) or modification (MOD),
- a is the author of the contribution,
- t is the timestamp of the contribution.

As mentioned in Section 2, we focus on the code that manages the instantiation of the runtime architecture of the software. When an architectural framework is used, explicit architecture descriptions can be characterized in the code by a set of distinctive markers (such as XML tags or Java annotations). Code contributions are thus split in two sets: those that define the runtime architecture — that will be called **architectural contributions** — and others — **non architectural contributions**.

3.2 Contributor Importance Measurement

In order to measure the importance of the contribution of a developer, several works have defined ownership metrics. Bird et al. (2011) consider that a developer contributes to the authorship of a code library to the extent of a ratio calculated as the number of his / her contributions over the total number of contributions made on this library. Foucault et al. (2014) have gone a step further by providing a formal model to compute such an ownership ratio.

In our work, the model proposed by Foucault et al. 2014 is adapted in two ways. Firstly, our proposed ownership metrics is **global** to (a version of) a project. This global scope better assesses contributions to the runtime architecture, as they are not local but span on the whole project. Secondly, it proposes an adaptation of this metrics to **architectural contributions**.

According to Foucault et al. (2014), the ownership ratio for a developer d on a software module m is defined as follows. Let:

- M be the set of modules of a given (version of a) software project,
- D be the set of developers for this (version of a) project,

We define:

- $w(m,d)$ as the number of contributions made by developer $d \in D$ on module $m \in M$
- $w(m)$ as the total number of contributions made on module $m \in M$ by all developers of D

The ownership ratio for developer d on module m computes as:

$$own(m,d) = \frac{w(m,d)}{w(m)}$$

Global Contribution Ownership. Firstly, to express a global ownership ratio for a contributor d on the whole (version of a) project, we define:

- $w(d)$ as the number of all contributions made by contributor d on the whole project.
- w_D as the number of all contributions made by all contributors in D on the whole project.

The global ownership of contributor d on the whole project computes as:

$$own(d) = \frac{w(d)}{w_D}$$

Minor and Major Contributors. As presented in Bird et al. (2011), contributors are then split into two categories, based on the proposed global ownership measure and on a 5% threshold. This threshold has been validated by Bird et al. (2011) with a sensitivity analysis. We checked that a threshold variation in the range 2 to 10 gives similar results. Contributors are considered **minor contributors** for a project version when their ownership ratio is lower or equal to 5%. They are conversely **major contributors** when their ownership ratio is strictly greater than 5%. In the remaining, minor contributors will be abbreviated *mCs* and major contributors *MCs*.

Global Runtime Architecture Code Ownership. We propose an original global runtime architecture code ownership measure derived from Foucault et al. (2014). We define:

- $w_a(d)$, as the number of runtime architecture code contributions made by contributor d on the whole project.
- w_{aD}, as the number of all runtime architecture code contributions made by all contributors D on the whole project.

The **global runtime architecture code ownership ratio** own_a for contributor d, computes as:

$$own_a(d) = \frac{w_a(d)}{w_{aD}}$$

Non / Minor / Major Runtime Architecture Code Contributors. Developers are split into ~~two~~ three categories regarding the importance of their contribution to the code of the runtime architecture. Non runtime architecture contributors have authored no line of code related to the definition of the runtime architecture up to the current analyzed version in the history of the project. Runtime architecture contributors are them split into two categories, major and minor, based on a 5% threshold. This threshold is chosen to be coherent with categories already calculated for code contributors.

In the remaining, non runtime architecture code contributors will be abbreviated *NRACs*, minor runtime architecture code contributors *mRACs* and major runtime architecture code contributors *MRACs*.

4 PROPOSED APPROACH

Project files are first extracted from the repository. The result is a raw set of code and deployment descriptor files for each analyzed version of the project. Two independent treatments are then executed for each file. First, a complete history of its modifications is retrieved, as a list of code line changes along with their authors and contents. Second, a syntactic analysis is realized as a transformation of the file content into an Abstract Syntax Tree (AST). Next step consists in mapping the AST nodes to their corresponding line changes in order to identify which contributions relate to the runtime architecture definition (semantic analysis). Contribution importance is then calculated for each developer.

This process has been implemented with the following technologies: Git as the project repository, Git-blame to retrieve file change histories, Java Development Tool (JDT) to parse code files or deployment descriptors and to handle the resulting ASTs and Python and R languages to extract contributor roles and run statistical analyses. None of these implementation choices is restrictive: these technologies are based on general principles and could either be used in other ecosystems (Git and Git-blame) or have some equivalent (Spring framework or JDT).

5 EMPIRICAL CASE STUDY

This section presents a case study run on the *BroadleafCommerce* open-source project retrieved
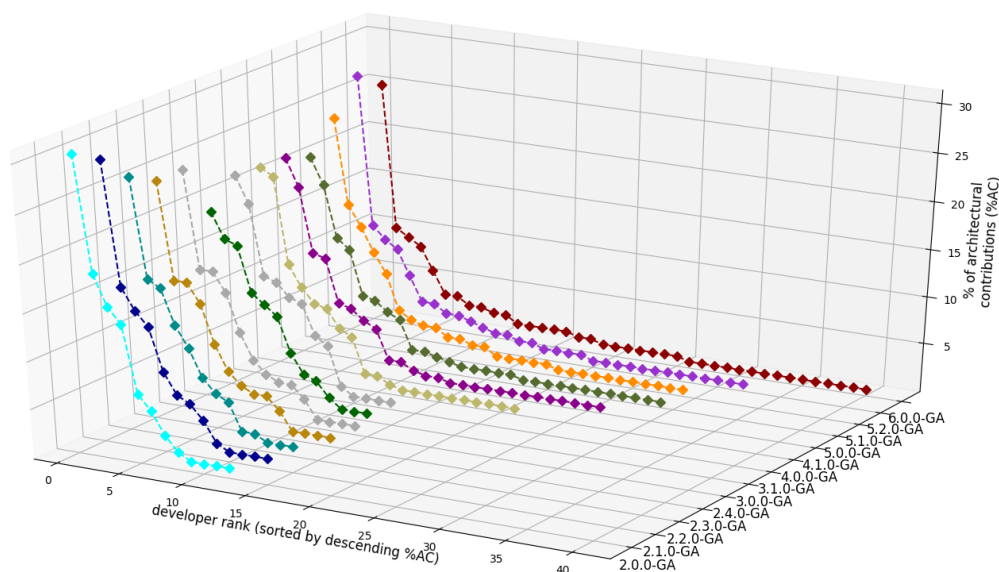
Figure 1: Percentages of contributions to runtime architecture by versions and by contributors.

from GitHub. This project has been selected for its use of the Java Spring Framework and its conformance with the criteria proposed in Jarczyk et al. (2014) to identify significant projects. Indeed, the *BroadleafCommerce* project has 1285 stars (greater than 100) and it has been forked 1036 times (greater than 10). These criteria characterize a corpus of 524 GitHub projects. *BroadleafCommerce* also has many contributors — over 70 in versions 6.X.Y — and a long history — 184 branches and 323 released versions. Finally, *BroadleafCommerce* is an open-source but industrial project, providing a framework for the development of professional e-commerce websites.

Using Git we have extracted 13 *General Availability* versions of *BroadleafCommerce* (from 2.0.0-GA to 6.0.0-GA) for analysis. These versions correspond to stable, production-ready versions developed over 7 years. Data used to perform empirical analysis are available online[2].

Answers to our proposed research questions are discussed in the following sections.

RQ₁: Can contributor profiles be mined from contributions to runtime application architecture definitions?

Figure 1 gives the percentages of contributions to the runtime architecture for each contributor and each version. To better show tendencies, contributors are sorted by their level of contributions for all versions. For all versions, the metrics clearly sepa-

rates three sets of contributors: developers that do not have any contribution to the runtime architecture (*i.e.*, NRACs which are actually not plotted), developers with scarce contributions (*i.e.*, *mRACs*, plotted on the flat right part of the curves) and developers with significant contributions (*i.e.*, *MRACs*, plotted on the higher left part of the curves). Moreover, Figure 1 graphically confirms that 5% is a relevant threshold to separate *mRACs* from *MRACs*.

We also measure global ownership ratios to separate major code contributors from minor code contributors. Figure 2 presents the evolution of the resulting contributor categories over the different analyzed versions. The metrics enable to separate the contributors in all possible categories combinations, except for the *NRAC-MC* category that do not seem to exist in the project. We observe that the *MRAC-MC* category is rather stable over time and that a significant number of contributors are (and remain) *mRACs* or *NRACs*.

These figures show that our proposed metrics, based on the ownership of the runtime architecture code, is relevant to profile contributors on this project. Although simple, it appears to be not over sensitive as it is able to detect *mRACs* and *NRACs* in every version, spanning over 7 years, and thus different contexts (from a small starting project to a large stable one). Its simplicity is probably balanced by the very specific nature of the code that is measured (our first hypothesis). Conversely, our metrics does not seem to be over specific, as the *MRAC* category is quite stable over the time and may correspond to a core team of developers with a specific level of responsibility regarding architecture code development.
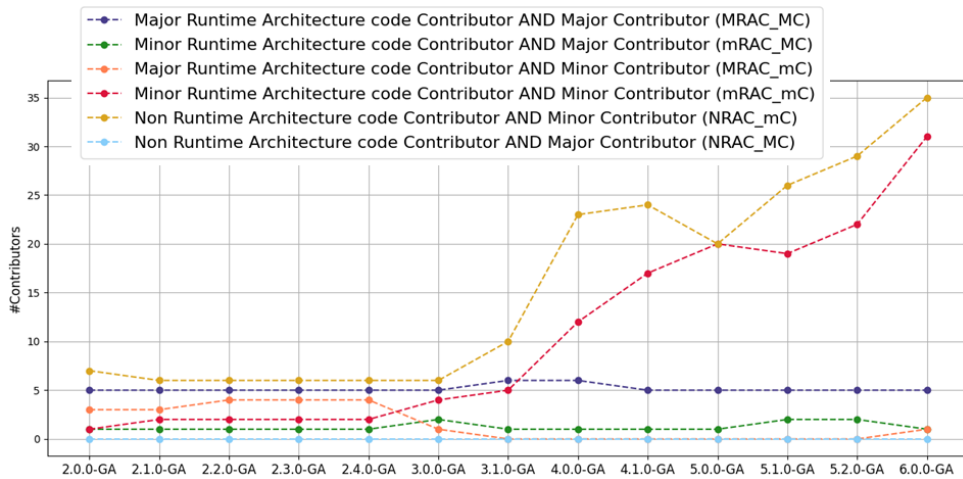
---

[2]https://anonymous.4open.science/r/
a390e5b2-fb34-4f2f-b8ed-d758c4430b8c/

Figure 2: Architectural contributions by categories of developers.

This is a promising proof of concept for $RQ1$. A full validation on a large set of projects is a near perspective for this work. This will also be an opportunity to further study and adapt the threshold values used in order to fine tune the separation between contributor categories.

RQ_2: Is there a link between runtime architecture and code contribution importance?

As our runtime architecture contribution metrics is based on specific contributions to the project code, it is necessary to study how these two kinds of contributions may be linked.

Figure 2 shows that *NRACs* are always *mCs*. However, the opposite implication is not true. Indeed, *mCs* are rather equally distributed between non contributors and contributors to the runtime architecture. *mCs* that are *MRACs* also exist significantly in early project versions (prior to version 3.0.0).

In the same way, *MCs* are mostly *MRACs*. As stated above, the opposite implication is not true for early project versions because of *MRACs* being *mCs*. To analyze the situation objectively, we conduct a statistical test on the following hypothesis:

- $\mathcal{H}_0$: *Being a MRAC is independent of being a mC or MC* (null hypothesis)
- $\mathcal{H}_1$: *Being a MRAC is dependent of being a mC or MC* (alternative hypothesis)

Fisher tests are performed with risk $\alpha = 0.05$ on the 13 analyzed versions of the *BroadleafCommerce* project. Table 1 presents the results. The test is significant when *p-value* $< \alpha$ (bold figures).

The test confirms a relation between *MCs* and *MRACs* after the project reaches version 3.1.0. For earlier versions, relation cannot be statistically assessed because of the significant *MRAC-mC* category, as compared to the size of the *MRAC-MC* category and of project size.

These results give positive answers to RQ_2. As expected, *MCs* are generally also *MRACs*. This seems natural as the most experienced and committed developers are expected to be in charge of the most sensitive concerns (our second hypothesis). But interestingly, *MRACs* may also significantly be *mCs* in some versions. Being a *MRAC* is therefore not a simple consequence of being a *MC*. On the contrary, it seems to be an actively chosen role, as shown by the constant existence of *NRACs* and the stability of the *MRAC-MC* category. This observation is confirmed by RQ_3.

RQ_3: What is the turnover within the different contributor categories?

Figure 2 highlights interesting evolution for contributor categories: the *MRAC-MC* category is stable over time; *MRAC-mC* category disappears and the *NRAC* category grows significantly after version 3.1.0. RQ_3 investigates the turnover between categories to better understand the dynamics of these phenomena. To do so, we observe contributor movements between categories for each analyzed project version. An extra category named *External* is introduced to take into account contributors who come in or go out of the project. Figure 3 shows the results as chord diagrams. A category is represented by an arc on the border circle. A category change is materialized by an arrow pointed from the source to the target category. The amount of changes is represented by the width of the link between two categories. The unchanged population in a category is represented as an inner colored region. As an example, between version 2-0-0-GA and 2-1-0-GA, a part of the *NRAC-mC* category mi-

Table 1: Results of Fisher test on 13 versions of the *BroadleafCommerce* project.

version	2.0.0-GA	2.1.0-GA	2.2.0-GA	2.3.0-GA	2.4.0-GA	3.0.0-GA	3.1.0-GA
p-value	1	0.54545	1	1	1	0.24242	**0.01515**

version	4.0.0-GA	4.1.0-GA	5.0.0-GA	5.1.0-GA	5.2.0-GA	6.0.0-GA
p-value	**4.5566^{-6}**	0.00017	9.12131^{-5}	0.00031	0.00017	6.99102^{-5}

grates to the *mRAC-mC* category. The *NRAC-MC* category is not plotted because its population is null in all releases as seen in Figure 2. Figure 3 shows a very low turnover on the *MRAC-MC* category. This reveals that its stable size comes from the stability of its group of contributors. Determining whether this results from an explicit policy is beyond the scope of this paper but is a relevant future work. Nonetheless, the few new members in this category come from the *MRAC-mC* or the *mRAC-MC* categories (*i.e.*, only experienced and noticeable contributors). This may confirm the existence of a policy for the management of the core team of the project and, as a consequence, for the contributors to the runtime architecture code (as an industrial project, *BroadleafCommerce* is very likely driven by a small professional team corresponding to the detected *MRAC-MC* category).

The main part of the *MRAC-mC* category becomes *mRAC-mCs* in version 3.0.0 and the remaining member of the category becomes a member of the *MRAC-MC* category in version 3.1.0. This may show that remaining a *MRAC* in the long term implies to be or become a *MC*. However, the stability and low turnover of the *MRAC-mC* category in the early versions, as long as its re-appearance in version 6.0.0, thanks to a contributor coming from the *MRAC-MC* category, may also suggest that very specialized code contributors, focused on the runtime architecture, are required in specific phases of a project (one more interesting question for future work).

Another observation is that all external contributors join mainly as *NRACs* and never as *MRACs*. Moreover, most *NRACs* never become *mRACs*.

All these observations confirm the relevance of our metrics. Indeed, it is able to separate the project members into three categories with stable cores. Of course, this metrics is not intended for measuring all the architectural contributions. We nonetheless expect as a futurework that it can be used to estimate the level of architectural quality management in projects.

6 THREATS TO VALIDITY

Construct Validity. Software architectures are managed at several structural levels (project modules, source code, runtime deployment / instantiation, etc.) and depend on technological or conceptual choices. Architectural contributions are variable and multifaceted. Our metrics detects contributors that work significantly on the runtime architecture definition. Our metrics can thus at least measure significant contributions to architecture development. As previously mentioned, this is a starting point for profiling code contributors, with the perspective of analyzing the management of architecture development.

Internal Validity. As our metrics is based on specific code ownership, measures could be biased by code contribution importance. This validity threat is addressed by *RQ*2. *RQ*2 shows that there is no systematic relation between the importance of contributions to code and to runtime architecture definition.

External Validity. For now, presented results are limited to a proof of concept validated on a unique project. To extract broad empirical knowledge, it is required to analyze a large number of projects that use the Spring framework but also others architectural frameworks (*e.g.*, Apache Struts, OSGi or JEE). A bias may also be induced by the chosen programming language and its ecosystem. To overcome this issue, projects programmed with languages other than Java (*e.g.*, Javascript or C++) could be analyzed.

Reliability. The presented results have been cross-checked, thanks to two separate implementations of the mining process. Despite the fact that remaining bugs cannot be be formally excluded, results have been consistently reproduced.

Statistical Threats. In our hypothesis test using Fisher's test, we use a risk α of 5%. Such a risk leaves 5% of chances to find identical results with random values. It is a commonly accepted value for such studies. Another issue concerns the considered versions of the *BroadleafCommerce* project. In order to limit the size and computation time of the experimental case study, we consider only sampled versions (those tagged as *General Availability*). This might constitute a bias but contribution measures are always
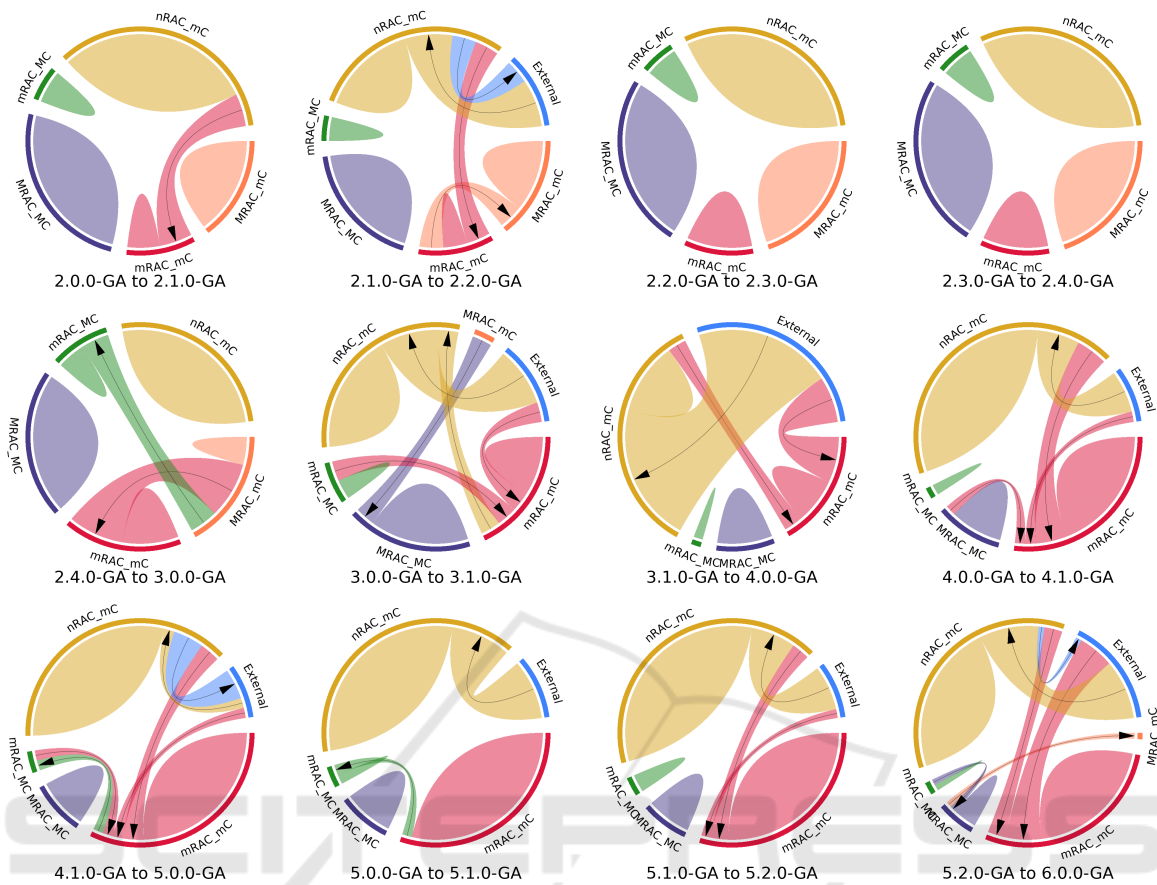
Figure 3: Chord diagrams illustrating changes in contributor categories between two successive versions.

calculated on complete histories of changes.

7 RELATED WORK

Expertise browser (Mockus and Herbsleb, 2002) is one of the earliest system created to find developer expertise from various data sources such as organization product list, source code, repository data and documentation. Three other approaches, (Sindhgatta, 2008; Schuler and Zimmermann, 2008; Teyton et al., 2014), use source code as their main data source to profile developers. Teyton et al. (2014) implements a domain specific language to model developer expertise. Other approaches do not use source code but data that indirectly qualify the code. Di Bella et al. (2013) propose a clustering approach based on software metrics and repository data to split developers in four groups according their contribution levels. Hauff and Gousios (2015) use GitHub metadata and ReadMe files to match job advertisements with developer profiles. CVExplorer (Greene and Fischer, 2016) uses the same inputs to produce a word list rep-

resenting the developer's domain of expertise. Only XTic (Teyton et al., 2014) and CVExplorer (Greene and Fischer, 2016) profile software architects.

Bird et al. (2011) are the first to use a contribution threshold to classify developers into two categories. Foucault et al. (2014) have conducted a study that uses the protocol of Bird et al. (2011) on open-source projects.

Developer turnover is a long standing problem that was first tackled in the 90's with a study on the impact of staff turnover on software project performance (Abdel-Hamid, 1992). Robles and Gonzalez-Barahona (2006) develop a deep study on turnover in large software projects. The impact of turnover on knowledge loss is measured by Izquierdo-Cortazar et al. (2009). Foucault et al. (2015a) also study the impact of developer turnover on software quality and mine contribution patterns that are correlated to turnover.

8 CONCLUSION AND PERSPECTIVES

This empirical analysis performed on the *Broadleaf-Commerce* project is a proof of concept for our proposed approach. It shows the existence of a core of major contributors to the runtime architecture definition that is globally stable over the studied history of the project and correlated with the existing stable core of major code contributors. Moreover, our metrics is precise enough to detect non and minor contributors to runtime architecture definition, that also form stable categories. As observed with our approach, runtime architecture development thus seem to obey management policies in the *BroadleafCommerce* project. This is very promising and opens many perspectives.

A first perspective is to conduct an empirical study on a large panel of projects. This would not only enable to fully validate our approach but also to characterize projects according to their architecture development management policies, as observed through our proposed metrics. Another perspective is to measure contributions to other architectural concerns and study complementarities and disparities between various architecture contributor profiles. In the same way, mapping contributions with architecture elements and structures would enable to study the existence of hotspots (areas that concentrate more contributions and contributors). The goal would be to advise the ideal number of contributors according to project size, chosen technology and contributor profiles.

REFERENCES

Abdel-Hamid, T. K. (1992). Investigating the impacts of managerial turnover / succession on software project performance. *Journal of Management Information Systems*, 9(2):127–144.

Abrahamsson, P., Babar, M. A., and Kruchten, P. (2010). Agility and architecture: Can they coexist? *IEEE Software*, 27(2):16–22.

Bird, C., Nagappan, N., Murphy, B., Gall, H., and Devanbu, P. (2011). Don't touch my code! examining the effects of ownership on software quality. In *19th ACM SIGSOFT FSE*, pages 4–14, Szeged, Hungary. ACM.

Booch, G. (1996). *Object solutions: managing the object-oriented project*. Addison-Wesley.

Di Bella, E., Sillitti, A., and Succi, G. (2013). A multivariate classification of open source developers. *Information Sciences*, 221:72–83.

Foucault, M., Falleri, J.-R., and Blanc, X. (2014). Code ownership in open-source software. In *18th EASE*, pages 1–9, London, UK. ACM.

Foucault, M., Palyart, M., Blanc, X., Murphy, G. C., and Falleri, J.-R. (2015a). Impact of developer turnover on quality in open-source software. In *10th ESEC / FSE*, pages 829–841, Bergamo, Italy. ACM.

Foucault, M., Teyton, C., Lo, D., Blanc, X., and Falleri, J.-R. (2015b). On the usefulness of ownership metrics in open-source software projects. *Inf. Softw. Technol.*, 64:102–112.

Garlan, D. and Shaw, M. (1993). *An Introduction to Software Architecture*, chapter 1, pages 1–39. on Software Engineering and Knowledge Engineering, vol. 2. World scientific.

Greene, G. J. and Fischer, B. (2016). CVExplorer: Identifying candidate developers by mining and exploring their open source contributions. In *31st IEEE/ACM ASE*, pages 804–809. ACM.

Gupta, P. and Govil, M. C. (2010). Spring Web MVC framework for rapid open source J2EE application development: a case study. *IJEST*, 2(6):1684–1689.

Hauff, C. and Gousios, G. (2015). Matching GitHub developer profiles to job advertisements. In *12th MSR*, pages 362–366, Florence, Italy. IEEE.

Izquierdo-Cortazar, D., Robles, G., Ortega, F., and Gonzalez-Barahona, J. M. (2009). Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *42nd HICSS*, page 10, Waikoloa, USA. IEEE.

Jarczyk, O., Gruszka, B., Jaroszewicz, S., Bukowski, L., and Wierzbicki, A. (2014). GitHub projects. quality analysis of open-source software. In *6th SocInfo*, pages 80–94, Barcelona, Spain. Springer.

Le Borgne, A., Delahaye, D., Huchard, M., Urtado, C., and Vauttier, S. (2018). Recovering three-level architectures from the code of open-source java Spring projects. In *30th SEKE*, pages 199–202, Redwood City, USA. KSI Research.

Mockus, A. and Herbsleb, J. D. (2002). Expertise browser: a quantitative approach to identifying expertise. In *24th ICSE*, pages 503–512, Orlando, USA. ACM.

Perez, Q., Le Borgne, A., Urtado, C., and Vauttier, S. (2019). An empirical study about software architecture configuration practices with the java spring framework. In *31st SEKE*, pages 465–593, Lisbon, Portugal. KSI Research.

Poncin, W., Serebrenik, A., and Van Den Brand, M. (2011). Process mining software repositories. In *15th CSMR*, pages 5–14, Oldenburg, Germany. IEEE.

Robles, G. and Gonzalez-Barahona, J. M. (2006). Contributor turnover in libre software projects. In *2nd IFIP OSS*, pages 273–286, Como, Italy. Springer.

Schuler, D. and Zimmermann, T. (2008). Mining usage expertise from version archives. In *5th MSR*, pages 121–124, Leipzig, Germany. ACM.

Sindhgatta, R. (2008). Identifying domain expertise of developers from source code. In *14th ACM SIGKDD KDD*, pages 981–989, Las Vegas, USA. ACM.

Taylor, R. N., Medvidovic, N., and Dashofy, E. (2009). *Software architecture: foundations, theory, and practice*. Wiley & Sons.

Teyton, C., Falleri, J.-R., and Blanc, X. (2012). Mining library migration graphs. In *19th WCRE*, pages 289–298, Kingston, Canada. IEEE.

Teyton, C., Palyart, M., Falleri, J.-R., Morandat, F., and Blanc, X. (2014). Automatic extraction of developer expertise. In *18th EASE*, page 8, London, UK. ACM.