Donatien Koulla Moulla[1], Alain Abran[2], Kolyang[3]

An Architecture for Effort Estimation of Solutions Based on Open Source

Abstract

Much software development is now conducted using other software products with open licenses and using a cooperative and distributed model. However, most research works on effort estimation of software projects have focused on conventional (traditional) projects with commercial licenses and are therefore not taking into account the software built using Open Source. This paper proposes an architecture for effort estimation of solutions based on Open Source.

Keywords: software effort estimation; Open Source estimation; estimation models; solutions based on Open Source

1. Introduction

Free/Open Source[4] software is playing an increasing role in information technologies in particular, and the world economy in general: much software development nowadays is conducted using other software products with open licenses and many are developed using a cooperative and distributed model. Many software companies rely on Open Source Software to develop their customer

[1] University of Ngaoundere, Department of Mathematics and Computer Science, Ngaoundere, Cameroon; University of Maroua, Institute of Mines and Petroleum Industries, Maroua, Cameroon, donatien.moulla@univ-ndere.cm

[2] École de Technologie Supérieure, Department of Software Engineering and Information Technology, Montréal, Canada, alain.abran@etsmtl.ca

[3] University of Maroua, The Higher Teachers' Training College, Maroua, Cameroon, kolyang@cde-saare.de

[4] The main difference between the free software and the Open Source movement remains essentially ideological. The Free Software movement is above all ethical and philosophical, based on knowledge sharing and mutual assistance, whereas the Open Source movement focuses on free software for their practical advantages. In this article, the term Free Software and Open Source will be used interchangeably.

solutions and products and Open Source Software enables companies to develop software systems at low cost[5]. However, most research works on effort estimation of software projects focus on conventional (traditional) projects based on proprietary, commercial licenses, and are therefore not taking into account software built using Open Source. Research on effort estimation for the development of Open Source software projects has become increasingly relevant due to the increasing number of organizations and governmental agencies for which Open Source software is included in their business strategy. Even though the differences between decentralized method of Open Source software development and software engineering practices have been debated[6], some aspects of Open Source development still need exploration. This provides the opportunity to compare the model of development based on Open Source with other models (traditional or proprietary)[7]. Because of the distributed and collaborative nature of Open Source software projects, the development effort invested in a project is usually unknown, even after the software has been released. Most of the approaches that have been developed for effort estimation of software development are based on proprietary development projects. The differences in the organization of work and the voluntary participation of developers in Open Source software projects justify why effort estimation of Open Source projects may differ. Indeed, Open Source estimation is different from the conventional one for several reasons: several assumptions of effort estimation models are inherently not relevant in the open source development context. For instance, COCOMO[8] assumes a good management by both the software producer and client, development following a waterfall-model and stability of the requirements during the whole process. However, in Open Source development, there is no distinction between the producer and client, and with regard to the other two assumptions (related to the model and requirements) the requirements are neither written down[9] nor constant over time, and Open Source software devel-

---

5 A. Ihara, A. Monden, K Matsumoto, *Industry Questions about Open Source Software in Business: Research Directions and Potential Answers*, 6th International Workshop on Empirical Software Engineering in Practice 2014, pp. 55–59.

6 S. Koch, *Effort Modelling and Programmer Participation in Open Source Software Projects*, Information Economics and Policy 2008.

7 Ibidem.

8 B.W. Boehm, *Software Engineering Economics* Prentice-Hall, New Jersey 1981.

9 P. Vixie, *Software Engineering*, in*: Open Sources: Voices from the Open Source Revolution*, eds. C. DiBona et al., O'Reilly, Cambridge 1999.

opment is closer to a spiral type of approach[10], often described as micro-spirals[11]. COCOMO II[12] on the other hand, does not contain these assumptions but incorporates a more prototype-oriented type of development. The Function Points sizing methods also do not contain any assumption concerning the process model as it is technology-independent and taking the user's viewpoint[13]. It is difficult to conduct an Open Source Software project following the traditional software development life-cycle models, such as the waterfall-model, because these models do not allow going back to a previous phase. In his book on Open Source Software development – "The Cathedral and the Baazar"[14] Eric S. Raymond defines the traditional development models[15] as the building of a cathedral with central planning, tight organization and one process from start to finish while the Open Source Software development model is defined as a *bazaar model* and described as "a great babbling bazaar of differing agendas and approaches". Table 1 presents a number of issues relevant for the comparison between the classical development and Open Source development.

These factors have an impact on the collection of effort data.

In spite of the significant importance of Open Source software for organizations and public administrations, there is a scarcity of research work on effort estimation in this domain. In the field of Open Source, there is a key difficulty related to research on estimation: while there are ample technical details available on the software itself (e.g. lines of code) there is little about the effort carried out to implement functionalities (which, most of the time, have not been measured). In Moulla et al.[16], other variables (attributes) are analysed in rela-

---

[10] B.W. Boehm, *A Spiral Model for Software Development and Enhancement*, "IEEE Computer" 1988, vol. 21, no. 5, pp. 61–72.

[11] T. Bollinger, R. Nelson, K.M. Self, S.J. Turnbull, *Open Source Methods: Peering through the Clutter*, "IEEE Software" 1999, vol. 16, no. 4, pp. 8–11.

[12] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Teece, *Software Cost Estimation with COCOMO II*, Prentice-Hall, New Jersey 2000.

[13] A.J. Albrecht, J.E. Gaffney, *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*, "IEEE Transactions on Software Engineering" 1983, vol. 9, no. 6, pp. 639–648.

[14] E.S. Raymond, *The Cathedral and the Bazaar*, O'reilly & Associates, Cambridge 1999.

[15] V. Tiwari, *Some Observations on Open Source Software Development on Software Engineering Perspectives*, "International Journal of Computer Science & Information Technology", December 2010, vol. 2, no. 6.

[16] D.K. Moulla, Kolyang, *COCOMO Model for Software Based on Open Source: Application to the Adaptation of TRIADE to the University System*, "International Journal on Computer Science and Engineering" 2013, vol. 5, pp. 522–527; D.K. Moulla, I. Damakoa, Kolyang, *Application of Function Points to Software Based on Open Source: A Case Study*, Proceedings of

tion with effort and, therefore, the cost (the cost is strongly related to effort) during the development of software solutions based on Open Source. The main cost driver in software development projects is typically the effort and here, like according to most researchers in this field of software effort estimation, the scope of this paper is limited to effort estimation (cost estimation is a significantly distinct topic with a lot of variables in addition to effort). More specifically, this paper proposes an architecture for effort estimation for software projects based on Open Source.

Table 1. Relevant Issues for the Comparison between the Estimation Context of a Classical Software Project and the Estimation Context of a Typical Open Source Project

Estimation context of a classical software project	Estimation context of a typical open source project
There is a customer who: – Pays for the software (therefore, he/she has some control over the resources available, on the planning and on penalties if the deliverables are not produced, or of poor quality – for instance, he/she can refuse to pay!). – States/documents his/her needs/requirements. – Will use the functionality. – Wants the software for a specific date.	**The customer:** – Not known in advance. The customer is some future random users. He/she can participate, or not, in the development of software. – Does not specify in advance the functionality he/she wants and at what level of detail. – Does not pay: therefore, he/she has no control over the project. – Coordination activity is done by core-developers.
There is a software development team: – With a project manager to plan and control (for the budget and delivery date). – Paid people (therefore, they are 'accountable' to their boss and customer). – Paid people record their effort and work time in the recording system to collect historical data for management purposes. – Classical effort estimation models are designed in this context.	**The Open Source Software developers' context:** – OSS may be supported by people all over the world. The developers are users of the software. They are not known in advance. Generally, there is no formal project management regime, budget or schedule. – The motivations to contribute to OSS development are: improve programming skills; the code for a project is intellectually stimulating to write, enhance professional status, enhance reputation in the OSS community, etc.

the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam 2014.

Estimation context of a classical software project	Estimation context of a typical open source project
	– The frequency of participation when all work is volunteered is influenced usually by work-related needs. – There is no formal project manager, so the customer or manager does not have any control over the volunteers. – There is no formal document for requirements. – Volunteers make some commitments and these commitments are usually monitored/recorded by a versioning system, mailing lists, a bug-tracking system, etc.

Source: the authors' own study.

This paper is structured as follows: section 2 presents the studies related to existing software effort estimation models, in particular to the modelling of the effort in Open Source software projects and those based on Open Source, and studies related to the adaptation of the software based on Open Source software and reused software. Section 3 presents the proposed architecture. Section 4 concludes the work and suggests directions for future work.

2. Related Works

This section presents the studies related to the existing software effort estimation models, especially to the modelling of the effort in Open Source software projects and those based on Open Source and, on the other hand, to the adaptation of the software based on Open Source software and software reuse.

The most known model in the effort estimation literature is probably the COCOMO model[17]. This model proposes an algorithmic formula for effort estimation based on software size, initially measured in lines of code. In response to practitioners' difficulties in estimating reasonably well the SLOC (Source Lines Of Code) before a project is well under way, new models have been developed that do not use SLOC as the primary input. In 1979, Allan Albrecht showed

[17] B.W. Boehm, *Software Engineering Economics* Prentice-Hall, New Jersey 1981.

his interest in general in the productivity measurement problem in the systems development, and created the function points method as an alternative to SLOC in estimation models[18]. There are now five different functional size measurement methods (COSMIC, IFPUG, NESMA, FiSMA and Mk II Function Points Analysis) recognized by the International Standards Organization (ISO).

It is interesting to identify and estimate before using OSS how much effort will be required to make changes and to integrate. There are many factors to be taken into account when choosing to develop on the basis of Open Source software, such as the size of the functionality to be implemented, software quality and software productivity. In the Open Source context, code quality is a fundamental design principle[19]: the quality is a direct consequence of architectural design decisions[20]. More open governance leads to higher design quality. More governance in Open Source projects increases the development effort[21]. However, in Open Source software estimation, maintenance effort is lower than in the proprietary development context due to a higher quality of code[22]. The maintenance effort of Open Source applications may not show a similar increasing trend over time[23].

In software development, productivity is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code or, preferably due to the independence from the programming language, in function points[24].

---

18 A.J. Albrecht, *Measuring Application Development Productivity*, IBM Application Development Symposium, Monterey, October 1979, pp. 14–17.

19 E. Capra, C. Francalanci, F. Merlo, *The Economics of Community Open Source Software Projects: An Empirical Analysis of Maintenance Effort*, "Advances in Software Engineering" 2010; I. Stamelos, L. Angelis, A. Oikonomou, G.L. Bleris, *Code Quality Analysis in Open Source Software Development*, "Information Systems Journal" 2002, vol. 12, no. 1, pp. 43–60.

20 J. Asundi, R. Kazman, M. Klein, *An Architectural Approach to Software Cost Modeling*, Second International Workshop on Economics-driven Software Engineering Research, Limerick 2000.

21 E. Capra, C. Francalanci, F. Merlo, *An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects*, "IEEE Transactions on Software Engineering" 2008, vol. 34, no. 6, pp.765–782.

22 P. Anbalagan, M. Vouk, *On Predicting the Time Taken to Correct Bug Reports in Open Source Projects*, Software Maintenance 2009, ICSM 2009, IEEE International Conference on, IEEE 2009, pp. 523–526; E. Capra, C. Francalanci, F. Merlo, *The Economics of Community Open Source Software Projects: An Empirical Analysis of Maintenance Effort*, "Advances in Software Engineering" 2010.

23 E. Capra, C. Francalanci, F. Merlo, *The Economics of Community Open Source Software Projects: An Empirical Analysis of Maintenance Effort*, "Advances in Software Engineering" 2010.

24 A.J. Albrecht, J.E. Gaffney, *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*, "IEEE Transactions on Software Engineering" 1983, vol. 9, no. 6, pp. 639–648.

Koch[25] has reported that in Open Source development projects, the distribution of effort between the participants (programmers) is skewed. According to Asundi[26], given the difficulty in managing resources in closed source projects, planning and delivering projects that are based on an Open Source community can be a bigger challenge. Resource allocation and budgeting will be harder and without a rigorous basis due to the distributed and collaborative nature of open source software projects. For this reason, he argues that using existing effort estimation models for Open Source projects has many disadvantages and thus, distinct effort estimation models are needed for this purpose. Robles et al.[27] present a new approach to estimate effort invested in an Open Source project by considering data from mining repositories of the source code. In their study, they proceed by the evaluation of developers' activity and their identification as full time developers, part time developers or occasional developers. The specific methodology they use to 'identify' as full-time, part-time or occasional consists in measuring for each developer:

- the number of commits merged in the code base during a given period;
- the number of active days during a given period, considered as days in which a developer performs at least one commit to the code base.

Their model is based on finding a threshold value t for the number of commits (or active days) for which they identify full-time, part-time and occasional developers with a minimum error. According to their Open Source estimation model, activity of all full-time developers would be above t while non full-time developers (part-time and occasional ones) would stay below t. In the proposed model framework, the number of commits is also taken into account. They show that their model offers a simple way of obtaining a software development estimate with bounded margins of error. Amor et al.[28] propose characterizing the activity of a developer through a versioning system, e-mails, a bugs tracking

---

[25] S. Koch, *Profiling an Open Source Ecology and its Programmers*, "Electronic Markets" 2004, vol. 14, no. 2, pp. 416–429.

[26] J. Asundi, *The Need for Effort Estimation Models for Open Source Software Projects*, 5-WOSSE Proceedings of the 5th Qorkshop on Open Source Software Engineering, New York 2005.

[27] G. Robles, J.M. González-Barahona, C. Cervigón, A. Capiluppi, D. Izquierdo-Cortázar, *Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack*, MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad 2014.

[28] J.J. Amor, G. Robles, J.M. González-Barahona, *Effort Estimation by Characterizing Developer Activity*, Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research, ACM 2006, pp. 3–6.

system etc. in order to measure the total effort invested in a project. This study has been repeated by Kalliamvakou et al.[29] to measure the contribution of developers from software repositories. Capiluppi et al.[30] tried to determine the average of hours work per day by the Linux kernel developers. Mockus et al.[31] have shown that the top 15 of nearly 400 programmers in the Apache project added 88 per cent of the total of lines of code. Mockus et al.[32] compared the productivity of the best developers of the Apache project and those of five commercial projects. They defined productivity as the mean number of Lines of Code per developer: this approach to measure productivity is rather fuzzy because productivity measurement depends on how it is defined, how it is measured and what are the assumptions and constraints. Moulla et al.[33] applied the COCOMO model and Function Points with real data from an Open Source Project, namely TRIADE version 7.a in order to show that the development of software based on Open Source has its advantages in terms of effort compared to the development from scratch: these findings illustrate that the use of Open Source software as a basis for further development can reduce effort and implementation time of a product (software). Further work is needed to explore whether what has been found in a rather small context with few data, can be extended to larger samples representing additional contexts and constraints.

[29] E. Kalliamvakou, E. Gousios, G. Spinellis, D. Pouloudi, *Measuring Developer Contribution from Software Repository Data*, MCIS 2009 4th Mediterranean Conference on Information Systems, Athens 2009, pp. 600–611.

[30] A. Capiluppi, D. Izquierdo-Cortazar, *Effort Estimation of FLOSS Projects: A Study of the Linux Kernel*, "Journal of Empirical Software Engineering" 2013, vol. 18, no. 1, pp. 60–88.

[31] A. Mockus, R. Fielding, J. Herbsleb, *Two Case Studies of Open Source Software Development: Apache and Mozilla*, "CM Transactions on Software Engineering and Methodology" 2002, vol. 11, no. 3, pp. 309–346.

[32] Ibidem.

[33] D.K. Moulla, Kolyang, *COCOMO Model for Software Based on Open Source: Application to the Adaptation of TRIADE to the University System*, "International Journal on Computer Science and Engineering" 2013, vol. 5, pp. 522–527; D.K. Moulla, I. Damakoa, Kolyang, *Application of Function Points to Software Based on Open Source: A Case Study*, Proceedings of the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam 2014.

3. A Proposed Architecture

In the Open Source software context, on the one hand, the large availability of heterogeneous software components on the internet allows developers to choose the most appropriate reusable components freely. On the other hand, it is not easy to find the most relevant Open Source software product for a particular business objective because of the huge number of existing Open Source products and their multiple versions. To find a relevant product, Ihara et al.[34] propose using software search engines such as SPARS, Kobers and Jarhoo. To find the appropriate version of a product, Mileva et al.[35] have proposed a library recommender system called AKTARI. In the software development process based on Open Source (Open Source components), steps of selection, analysis and testing of software components may be time consuming. When a developer finds a bug or wants to add a new functionality to an Open Source product, debugging and adding functionality is usually very difficult because the code has been developed by someone else in the OSS community. To solve these problems, several techniques have been proposed[36].

Wieringa et al. have identified six strategies for generalizing software engineering theories[37]. The present work is a case-based research. In case-based research, the variability of the real world is reduced by decomposing a case into components that can produce case phenomena by their interactions[38]. This study comes from the phenomenon observed by Moulla et al. in their case study[39].

---

[34] A. Ihara, A. Monden, K. Matsumoto, *Industry Questions about Open Source Software in Business: Research Directions and Potential Answers*, 6th International Workshop on Empirical Software Engineering in Practice 2014, pp. 55–59.

[35] Y.M. Mielva, V. Dallmeier, M. Burger, A. Zeller, *Mining Trends of Library Usage*, In Proceedings of the International Workshop on Principles of Software Evolution 2009, pp. 57–62.

[36] K. Dongsun, T. Yida, K. Sunghun, Z. Andreas, *Where Should We Fix This Bug? A Two Phase Recommendation Model*, "IEEE Transactions on Software Engineering" 2013, vol. 39, issue 11, pp. 1597–1610; D. Beyer, *Co-change Visualization Applied to PostgreSQL and ArgoUML*, In Proceedings of the 3rd International Workshop on Mining Software Repositories 2006, pp. 165–166; B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, *Feature Location in Source Code: A Taxonomy and Survey*, "Journal of Software: Evolution and Process" 2013, vol. 25, no. 1, pp. 53–95.

[37] R. Wieringa, M. Daneva, *Six Strategies for Generalizing Software Engineering Theories*, "Science of Computer Programming" 2015, no. 101, pp. 136–152.

[38] Ibidem.

[39] D.K. Moulla, Kolyang, *COCOMO Model for Software Based on Open Source: Application to the Adaptation of TRIADE to the University System*, "International Journal on Computer Science and Engineering" 2013, vol. 5, pp. 522–527; D.K. Moulla, I. Damakoa, Kolyang,

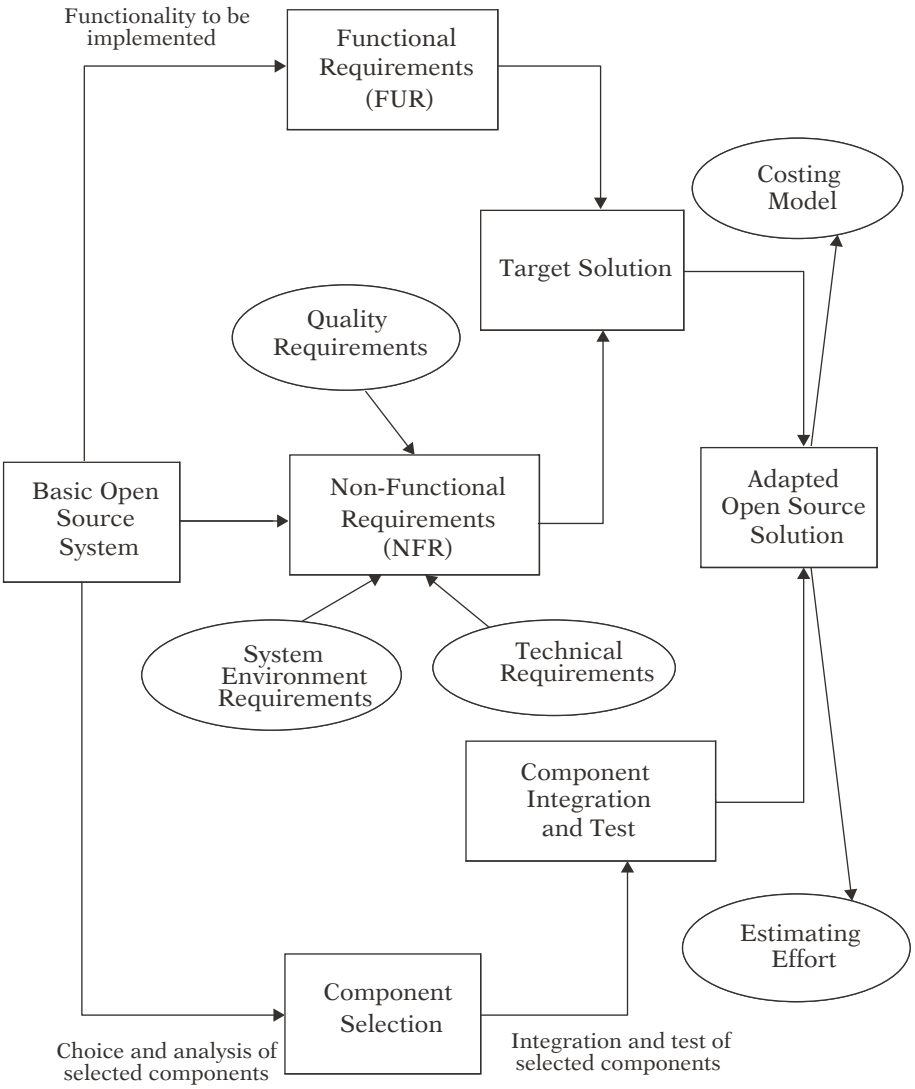

Figure 1. Proposed Architecture (Prototype) to Estimate Effort of a Solution Based on Open Source

Source: the authors' own study.

They analysed the development of the software (solution) based on Open Source Software called TRIADE. The development was done by analysing

Application of Function Points to Software Based on Open Source: A Case Study, Proceedings of the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam 2014.

both the source code (complexity of code) and system functionalities (functional requirements, non-functional requirements, etc.). Our architecture can be applied to similar cases (projects) and is intended both for practitioners and the industry. To conduct the adaptation of a software system, re-engineering is recommended. To understand how the system is built, usually two main sources of information can be used[40]:

- the source code of the application;
- the users of the platform.

The starting point to understand how the system is built is to understand how the different users interact with the system (system functionalities). Indeed, the way users interact with the system helps to understand how the system is used from the business point of view (system functionalities or Functional User Requirements – FUR). In addition, an analysis of the source code of software system to adapt and its execution in production help to retrieve the architecture of an application with its different components and classes (code complexity).

The basic Open Source system contains pieces of software (components) that need to be adapted. The Functional Requirements describe what the solution should do (i.e. the functionality to be implemented). ISO 14143–1[41] defines the Functional Requirements as a sub-set of the user requirements (i.e., requirements that describe what the software shall do, in terms of tasks and services).

The Non Functional Requirements describe how well the solution performs its task. The Non Functional Requirements characterize the software constraints which include quality and technical requirements (maintainability, portability, security, reliability, performance, documentation, etc.). ISO 24765[42] defines a Non-Functional Requirement as a software requirement that describes not what the software will do but how the software will do it. According to COSMIC[43] and IFPUG[44] definition, a Non-Functional Requirement is any requirement for the software part of a hardware/software system or of a software product, including how it should be developed and maintained, and how

---

[40] D.K. Moulla, I. Damakoa, Kolyang, *Application of Function Points to Software Based on Open Source: A Case Study*, Proceedings of the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam 2014.

[41] ISO/IEC 14143/1:2011, *Information Technology = Software Measurement – Functional Size Measurement*.

[42] ISO/IEC 24765:2010, *Systems and Software Engineering Vocabulary*.

[43] 'COSMIC' = the Common Software Measurement International Consortium (www.cosmic-sizing.org).

[44] 'IFPUG' = the International Function Point Users Group (www.ifpug.org).

it should perform in operation, except any functional user requirement for the software[45]. Non-functional requirements concern:

- the software quality;
- the environment in which the software must be implemented and which it must serve;
- the processes and technology to be used to develop and maintain the software and the technology to be used for the software execution, etc. (the technical aspects).

Figure 2 presents the main classes of Non-Functional Requirements.

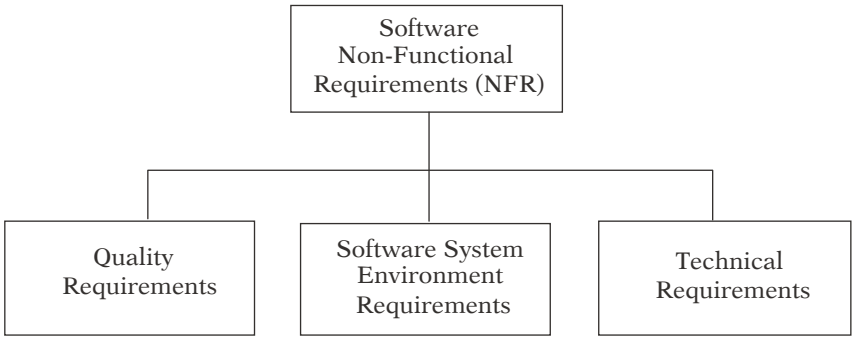

Figure 2. Summary Model of Software Non-Functional Requirements

Source: the authors' own study.

According to COSMIC and IFPUG, quality requirements are defined as requirements for the quality or for the architecture or design of the delivered system or software product[46].

System environment requirements are the characteristics of the environment in which the software system is developed and maintained and which it must support in operation[47]. Technical requirements are the requirements for how the software will be built, such as the programming language to be used and for the technology (hardware and communications) that the system will need in operation[48].

Based on these requirements, a target solution is designed. The effort involved in each activity can then be estimated by taking into account all the attributes

[45] COSMIC/IFPUG *Glossary of Non-Functional Requirements and Project Terms*, v. 1.0, June 2015.

[46] Ibidem.

[47] Ibidem.

[48] Ibidem.

in relation with effort. The costing model can be derived from the target solution, schedule and adapted Open Source solution.

Some studies have been done on effort modelling and participation of developers in Open Source Projects[49]. For solutions based on Open Source, the effort involved in each activity must be estimated. The effort to identify and estimate is indeed the effort to make changes and to integrate since the basic Open Source system is already developed. The total effort invested is the sum of effort involved in each activity.

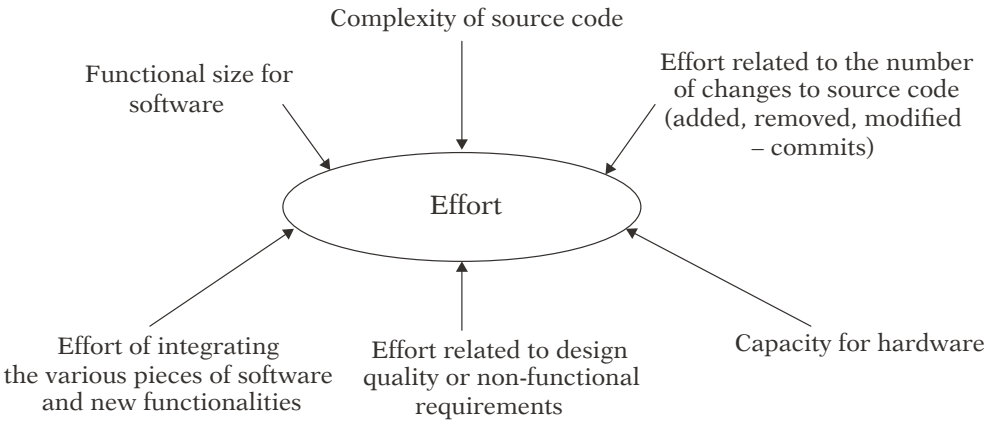

Figure 3. Schema on Tasks-related Effort Estimation for Solutions Based on Open Source

Source: the authors' own study.

In the context of solutions based on Open Source, all attributes or factors in relation with effort must be also taken into account, such as:

- the effort of integrating the various pieces of software: it represents effort required to integrate the pieces of given software;
- effort related to quality attributes or non-functional requirements;
- the number of changes to source code (commits): it represents the amount of work devoted to make changes;
- effort related to the complexity of code: it represents effort required to understand how the source code is built; the particularity of the Open Source

[49] S. Koch, *Effort Modelling and Programmer Participation in Open Source Software Projects*, Information Economics and Policy 2008; S. Koch, G. Schneider, *Effort, Cooperation and Coordination in an Open Source Software Project: GNOME*, "Information Systems Journal" 2002; S. Koch, *Organisation of Work in Open Source Projects: Expended Effort and Efficiency*, "Revue d'économie industrielle" 2011, no. 136, 4ème trimester.

software as it relates to effort estimation has to do with the fact that developers are required to work on a completely unfamiliar code;
- capacity for hardware;
- functional size for the software.

The architecture represents how the non-functional are addressed in the solution.

4. Discussion and Future Work

This paper addressed the issue of an architecture of effort estimation for solutions based on Open Source. Most research works on effort estimation of software projects have focused on conventional (traditional) projects with commercial licenses and are therefore not taking into account the software built using Open Source. An estimation architecture has been proposed for this purpose. Compared to previous studies related to effort estimation, the proposed architecture seems to be close to these ones. Like some existing effort estimation models, the proposed framework takes into account functional requirements, non-functional requirements, capacity for hardware etc. However, our proposal includes, in addition, the effort of integrating the various pieces of software and new functionalities and effort related to the number of changes to the source code (added, removed, modified – commits).

Architectures of estimation models often make assumptions about the context in which they are intended to be applied. As such, the proposed architecture presented here has some limitations. For instance, this architecture must take into account all attributes in relation with effort and has to be validated by software estimation experts.

The present study constitutes an exploratory research frame on the development of specialized effort estimation models for solutions based on Open Source.

We plan to refine our architecture as part of our research in the near future. We will take into account other attributes in relation with effort which have not been mentioned in this article. We plan to conduct a more detailed discussion of what effort/cost drivers are missing from (or need a reinterpretation for) the traditional estimation models that are needed to successfully estimate Open Source development effort. This includes how to evaluate the proposal in the real-life context and how to make generalizations a more robust analytical induction as part of our research in the near future.

References

Albrecht A.J., Gaffney J.E., *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*, "IEEE Transactions on Software Engineering" 1983, vol. 9, no. 6, pp. 639–648.

Albrecht A.J., *Measuring Application Development Productivity*, IBM Application Development Symposium, Monterey, October 1979, pp. 14–17.

Amor J.J., Robles G., González-Barahona J.M., *Effort Estimation by Characterizing Developer Activity*, Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research, ACM 2006, pp. 3–6.

Anbalagan P., Vouk M., *On Predicting the Time Taken to Correct Bug Reports in Open Source Projects*, Software Maintenance 2009, ICSM 2009, IEEE International Conference on, IEEE 2009, pp. 523–526.

Asundi J., Kazman R., Klein M., *An Architectural Approach to Software Cost Modeling*, Second International Workshop on Economics-driven Software Engineering Research, Limerick 2000.

Asundi J., *The Need for Effort Estimation Models for Open Source Software Projects*, 5-WOSSE Proceedings of the 5th Qorkshop on Open Source Software Engineering, New York 2005.

Beyer D., *Co-change Visualization Applied to PostgreSQL and ArgoUML*, In Proceedings of the 3rd International Workshop on Mining Software Repositories 2006, pp.165–166.

Boehm B. W, Abts C., Brown A.W., Chulani S., Clark B.K., Horowitz E., Madachy R., Reifer D.J., Teece B., *Software Cost Estimation with COCOMO II*, Prentice-Hall, New Jersey 2000.

Boehm B.W., *A Spiral Model for Software Development and Enhancement*, "IEEE Computer" 1988, vol. 21, no. 5, pp. 61–72.

Boehm B.W., *Software Engineering Economics* Prentice-Hall, New Jersey 1981.

Bollinger T., Nelson R., Self K.M., Turnbull S.J., *Open Source Methods: Peering through the Clutter*, "IEEE Software" 1999, vol. 16, no. 4, pp. 8–11.

Capiluppi A., Izquierdo-Cortazar D., *Effort Estimation of FLOSS Projects: A Study of the Linux Kernel*, "Journal of Empirical Software Engineering" 2013, vol. 18, no. 1, pp. 60–88.

Capra E., Francalanci C., Merlo F., *An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects*, "IEEE Transactions on Software Engineering" 2008, vol. 34, no. 6, pp.765–782.

Capra E., Francalanci C., Merlo F., *The Economics of Community Open Source Software Projects: An Empirical Analysis of Maintenance Effort*, "Advances in Software Engineering" 2010.

COSMIC/IFPUG *Glossary of Non-Functional Requirements and Project Terms*, v. 1.0, June 2015.

Dit B., Revelle M., Gethers M., Poshyvanyk D., *Feature Location in Source Code: A Taxonomy and Survey*, "Journal of Software: Evolution and Process" 2013, vol. 25, no. 1, pp. 53–95.

Dongsun K., Yida T., Sunghun K., Andreas Z., *Where Should We Fix This Bug? A Two Phase Recommendation Model*, "IEEE Transactions on Software Engineering" 2013, vol. 39, issue 11, pp. 1597–1610.

Ihara A., Monden A., Matsumoto K., *Industry Questions about Open Source Software in Business: Research Directions and Potential Answers*, 6th International Workshop on Empirical Software Engineering in Practice 2014, pp. 55–59.

ISO/IEC 14143/1:2011, *Information Technology = Software Measurement – Functional Size Measurement*.

ISO/IEC 24765:2010, *Systems and Software Engineering Vocabulary*.

Kalliamvakou E., Gousios E., Spinellis G., Pouloudi D., *Measuring Developer Contribution from Software Repository Data*, MCIS 2009 4th Mediterranean Conference on Information Systems, Athens 2009, pp. 600–611.

Koch S., *Effort Modelling and Programmer Participation in Open Source Software Projects*, Information Economics and Policy 2008.

Koch S., *Organisation of Work in Open Source Projects: Expended Effort and Efficiency*, "Revue d'économie industrielle" 2011, no. 136, 4ème trimester.

Koch S., *Profiling an Open Source Ecology and its Programmers*, "Electronic Markets" 2004, vol. 14, no. 2, pp. 416–429.

Koch S., Schneider G. Effort, *Cooperation and Coordination in an Open Source Software Project: GNOME*, "Information Systems Journal" 2002.

Mielva Y.M., Dallmeier V., Burger M., Zeller A., *Mining Trends of Library Usage*, In Proceedings of the International Workshop on Principles of Software Evolution 2009, pp. 57–62.

Mockus A., Fielding R., Herbsleb J., *Two Case Studies of Open Source Software Development: Apache and Mozilla*, "CM Transactions on Software Engineering and Methodology" 2002, vol. 11, no. 3, pp. 309–346.

Moulla D.K., Damakoa I., Kolyang, *Application of Function Points to Software Based on Open Source: A Case Study*, Proceedings of the Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, Rotterdam 2014.

Moulla D.K., Kolyang, *COCOMO Model for Software Based on Open Source: Application to the Adaptation of TRIADE to the University System*, "International Journal on Computer Science and Engineering" 2013, vol. 5, pp. 522–527.

Raymond E.S., *The Cathedral and the Bazaar*, O'Reilly & Associates, Cambridge 1999.

Robles G., González-Barahona J.M., Cervigón C., Capiluppi A., Izquierdo-Cortázar D., *Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack*, MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad 2014.

Stamelos I., Angelis L., Oikonomou A., Bleris G.L., *Code Quality Analysis in Open Source Software Development*, "Information Systems Journal" 2002, vol. 12, no. 1, pp. 43–60.

Tiwari V., *Some Observations on Open Source Software Development on Software Engineering Perspectives*, "International Journal of Computer Science & Information Technology", December 2010, vol. 2, no. 6.

Vixie P., *Software Engineering*, in*: Open Sources: Voices from the Open Source Revolution*, eds. C. DiBona et al., O'Reilly, Cambridge 1999.

Wieringa R., Daneva M., *Six Strategies for Generalizing Software Engineering Theories*, "Science of Computer Programming" 2015, no. 101, pp. 136–152.