MAESTROS: Multi-Agent Simulation of Rework in Open Source Software

Thiago R. P. M. Rúbio, Henrique Lopes Cardoso and Eugénio da Costa Oliveira

Abstract Rework Management in software development is a challenging and complex issue. Defined as the effort spent to re-do some work, rework implies big costs given the fact that the time spent on rework does not count to the improvement of the project. Predicting and controlling rework causes is a valuable asset for companies, which maintain closed policies on choosing team members and assigning activities to developers. However, a trending growth in development consists in Open Source Software (OSS) projects. This is a totally new and diverse environment, in the sense that not only the projects but also their resources, e.g., developers change dynamically. There is no guarantee that developers will follow the same methodologies and quality policies as in a traditional and closed project. In such world, identifying rework causes is a necessary step to reduce project costs and to help project managers to better define their strategies. We observed that in real OSS projects there are no fixed team, but instead, developers assume some kind of auction in which the activities are assigned to the most interested and less-cost developer. This lead us to think that a more complex auctioning mechanism should not only model the task allocation problem, but also consider some other factors related to rework causes. By doing this, we could optimise the task allocation, improving the development of the project and reducing rework. In this paper we presented MAESTROS, a Multi-Agent System that implements an auction mechanism for simulating task allocation in OSS. Experiments were conducted to measure costs and rework with different project characteristics. We analysed the impact of introducing a Q-learning reinforcement algorithm on reducing costs and rework. Our findings correspond to

T.R.P.M. Rúbio (✉) · H.L. Cardoso · E. da Costa Oliveira
LIACC / DEI, Faculdade de Engenharia, Universidade Do Porto,
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
e-mail: reis.thiago@fe.up.pt

H.L. Cardoso
e-mail: hlc@fe.up.pt

E. da Costa Oliveira
e-mail: eco@fe.up.pt

P. Novais et al. (eds.), *Intelligent Distributed Computing IV*,
Studies in Computational Intelligence 616,
DOI 10.1007/978-3-319-25017-5_6

a reduction of 31 % in costs and 11 % in rework when compared with the simple approach. Improvements to MAESTROS include real projects data analysis and a real-time mechanism to support Project Management decisions.

1 Introduction

Software development is facing a big change. Traditionally, software development companies have closed policies on choosing developer teams and rigorous control over the task allocation between them. In the last few years we have seen a massive adoption of Open Source Software (OSS) [1], based on free code in which any other developer can contribute by free will. Open source developers could be anywhere in the world, have learnt different techniques and ways of working. There is no guarantee that they will follow the same strategies and quality policies when compared with a traditional and closed project.

Resolving problems that were not solved consistently or bugs created by developer mistakes is called *Rework*, an additional and not planned work represented by the effort cost (in time and money) spent in order to resolve a problem with a requirement that was previously considered solved. Rework occur in both closed and open projects, but closed projects have many ways to early identify and control rework causes that open projects do not. To be able to manage rework in OSS is a necessary and important step to reduce project costs and to help project managers to better define their strategies to software improvement [2].

When considering open projects, a big problem is how to identify the main causes of rework. Developers introduce rework due to lack of specification, missing verifications or even unplanned new properties. In the literature, rework is considered a manifestation of the lack of communication between developers and a cause of stressed or uncommitted personnel [3]. Finding rework causes could help us to discover how developer's behaviours affect projects.

We observed that frequently activities are assigned to developers that are interested. There are no fixed team and the assignment of the activities rely only on developer's cost (mostly in time or even monetary) of development. Actually, this situation resembles a simple auction mechanism in which the activities are allocated to the less costly developers. A more appropriate mechanism would consider also other impacting factors such as developer experience and its past results on achieving activities completion with success. This motivates our work: we aimed at simulating this task allocation process using a MAS in order to understand, analyse and propose some optimisation that could help to reduce rework.

In this paper we present a multi-agent system for simulating the task allocation process in open source software. We seek the best opportunities: low development cost with minimum rework. We have evaluated the performance of our model in terms of the number of re-incident activities and their rework cost together with project final cost. Results show that our approach can get close-to-budget projects

final cost, even with some rework present. Further developments of our system could be a good ally to project managers.

The rest of this paper is structured as follows: In Sect. 2 we present the Rework problem in Open Source Software. Section 3 discusses the project management process workflow. In Sect. 4 we describe the architectural design of our system. Section 5 presents the experimental evaluation of the model. We discuss the findings of this work and point lines of future research in Sect. 6.

2 Related Work

Open Source Software (OSS) is a trend in software development. Since late 1990, an uncountable number of projects have grown in this environment proving it can be successful and profitable [4]. Research interest in OSS is much diversified [5]. The open nature of the software creates a great difficulty in managing resources, planning and delivering projects. As mentioned by Raja et al. [6], resource allocation and budgeting in OSS is even a harder challenge. The cost of the development is an important factor for the success of a project, making the search for reducing rework an important matter. In Sect. 1, we introduced rework as the effort and consequently monetary cost of trying to fix something that was already considered a solved problem. Rework is, in fact, a big problem in software engineering, consuming big part of the project budget (40 % up to 70 %) [7]. Rework could be explained by human problems in project management like communication, formation and work conditions [3, 7] and the Industry believes that great part of rework could be early identified and avoided, but until now not much attention has been paid in studying rework.

Previous works have characterised the relation between rework and developers actions regarding their expertise and work profiles. Rbio et al. [8] divided developers into members and volunteers, in which the first are recognised by Project Owners because of their knowledge on some specific project or are permanent members of a development team. Volunteers, on the other hand are developers that might contribute spontaneously by their interest on the project. Members are usually have a lower probability of generating rework (about 10 %), while volunteers have a higher chance (30 %). By other side, the development cost of a member is known to be more than volunteers work [8, 9]. The other actor interested in this process is the Project Manager (PM). Although Project Managers work under different methodologies, their basic task is to distribute projects activities and manage the assignment of tasks to the available developers [10].

In traditional development the tasks are imposed, opposing to more flexible methodologies, where developers are able to discuss or vote their willingness for working in some task [11]. As referred in Sect. 1, process of task assignment in OSS is somewhere between this two: the Project Manager tries to choose the developer that best fits to some activity by its reputation, cost and availability [12].

Each actor in this system has its own decisions: developers must decide whether they are interested on developing a specific activity, determine how their assigned

tasks will be accomplished, and even decide on which bids to propose. On the other hand, managers also make decisions in their effort on trying to reduce reworks and costs, while maximising the number of activities successfully concluded. This underlying autonomy of the actors involved lead us to an agent-based approach. Each software agent represents one actor in the process, behaving accordingly to its own goals. In multi-agent systems the autonomous entities (agents) can decide whether or not to accomplish some task and can deal with self and community goals [13].

The importance of mapping developers as agents relies on their free will to contribute to projects and the relationships built from their interactions. In fact, although many investigations about open source deal with some properties of OSS, like the actors roles in [14], very few discussions in the literature model the development process in this environment as a multi-agent system [15, 16].

Simulation is a good approach in this case, where the difficulty of gathering and analysing data on-line with real open source projects is high. Since the platforms restrain the access to data and most projects decisions are private, analysing the task allocation in real-time is hard and complex. The simulation, in the other side, needs to represent well the behaviours and the mechanisms used by agents to coordinate their actions. Once the abstracted characteristics represent significantly the behaviour of the actors, a multi-agent simulation could be used to investigate the impact of changing various characteristics of the project, for instance, how would PM behave when the number of available developers grow or even how to automate the negotiations about costs where the agents represent the interests of the real developers.

We do not intend to create a new method for optimising task allocation, but rather applying automated scheduling and negotiation intelligent techniques to help on the improvement of the decisions taken by project managers in open source projects. our investigation contributes by creating a first attempt to model the task assignment process in OSS with a MAS. Moreover, using learning strategies could lead to reduce the rework on this kind of projects.

As an introductory work, this opens opportunities to future works in this area and widens the applicability of MAS to a growing and rich environment.

3 Model Conceptualisation

The OSS development process described on the previous sections give us an insight about how to model actors and their behaviours. Transcribing agents goals, actions and decisions could help us to create a simulation model that represent how task assignment work on OSS projects [15].

Rubio et al. [8], analysed real data from big real open source projects such as Apache projects.[1] For simplicity, we are going to consider only the types of developers described in their work: members and volunteers. Regardless of its type, a developer can finish its tasks successfully or not. When the conclusion of an activity

[1] Apache Software Foundation—http://www.apache.org/.

fails, the Project Manager must reassign it causing an increase of cost, the so-called rework. When the current set of activities is successfully concluded, the Project Manager reports it to the Project Owner (PO), the person that knows the next milestones on the project or new set of activities to deliver to the development team. We characterise three main agents in the OSS environment: (1) Project Manager (PM), (2) Developer and (3) Project Owner. Developers are also divided into two subgroups: (a) Members and (b) Volunteers.

The Project Manager has the knowledge about a set of activities and the estimated cost for completing each one of them. The Project Manager must gather information about the interested developers and assign the activities to the best opportunities, considering risk, cost and other factors. Similar to an auction protocol, the PM wants to "sell" the activities to developers, who "bid" with the cost of development.

PM is, then, responsible to select the best developer for each activity. The complete process is represented in Fig. 1 and starts with the PM checking if there are to-do activities and selects one of them (based on priority or cost, for example). The information about the activity is spread among the developers who are not currently working and in case they are interested on developing such activity, they must send a proposal (cost of work) to the PM. For members this is mandatory and they must always present proposals. Volunteers could refuse to work on some activities. The PM evaluates the proposals received and if there are none, the activity goes back to the to-do list and the process starts again. On the other hand, if there are proposals, the PM awards the winner and sends the refusing message to the other bidders. At this time the winner developer starts the development process. This developer will not participate in other auctions until current work is done. The PM repeats the cycle with other activities until there are no more activities or available developers. Finally, when the developer finishes its work, it becomes available again by notifying the PM, who is responsible to check whether the activity was successfully completed or not. In case the developer has failed, the activity is put back into the to-do list. When the developers work is well done, the activity is put in the completed list. The process finishes when all the activities are completed.

In order to simulate this environment and analyse how the assignment process could be optimised in terms of the rework introduced and process completion, we

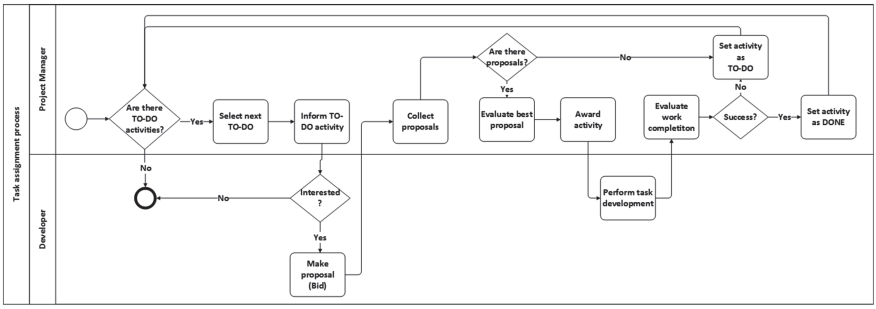

Fig. 1 Task assignment process

have proposed a multi-agent simulation. We analyse the problem contemplating the decisions taken by the Process Manager and propose the use of a reinforcement learning algorithm in order to reduce rework. When compared to a simple approach that only consider the available bids, the Q-Learning approach uses a more complex reasoning mechanism, by considering also the experience and past results from developers. Section 4 presents our model characteristics and architecture.

4 MAESTROS Architecture

We have analysed how Project Managers can distribute project tasks between available developers. As explained, work in on-line real projects is a complex matter. In order to study how the behaviours of the agents and how project manager decisions affect project results, we have created a simulation system that implements the process described on Sect. 3. Although there are many agent-based simulation platforms available, we wanted to create a flexible system that allows us to expand our work in the future, connecting the simulation with the real on-line project. Thus, we have constructed MAESTROS (**M**ulti **A**g**E**nt **S**imulation of **R**ework on **O**pen **S**ource Software).

Developed in Java, MAESTROS use JADE[2] multi-agent framework [17] in its core. JADE allows us to define agent behaviours that map actors reactions in different steps of the task assignment process.

We assume that the development cost of an activity is directly proportional to the time in which a developer is working on it. Basically, this simplifies the modelling of the inner development process of one activity into a time-frame cost problem. In our system, the cost of an activity is the portion of the time the developer will be occupied working on it. We do not simulate the working process described by one activity like coding generation or other kind of documentation by developers. Although this may seem a simplistic model, the focus here is whether the choices made could be optimised in order to reduce rework.

MAESTROS consists in three main parts: (1) Communication layer; (2) Negotiation mechanism and (3) Decision strategy. Regarding to the Communication layer, MAESTROS agents communicate through messages, sending messages with the desired content and the semantics of the information. For this, JADE We have used FIPA ACL messages [18], native on JADE platform. ACL messages uses standardised *performatives*, a special field on the message that contextualises the required action.

In the Negotiation Mechanism we model how the task allocation occurs. As explained, our model is a very simple auction mechanism. The Contract Net Protocol [19] is a good strategy that fits just our needs. The algorithm consists on one round of bidding in order to select the winner that will get the item or service auctioned. Activities could be seen as the services the Project Manager wants to sell

[2] http://jade.tilab.com/.

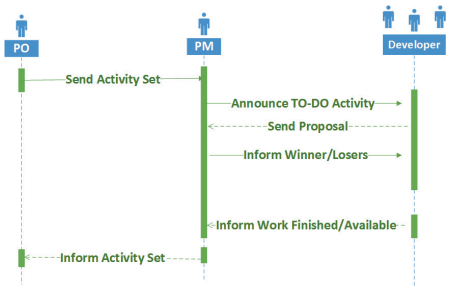

Fig. 2 Communication on the assignment process cycle

and developers are the buyers that bid (make a proposal) in terms of working cost of an activity. In fact, the Project Manager just needs one round to decide who will be responsible for executing the activity, assuring the Contract Net as a good strategy to MAESTROS.

Our implementation of the protocol follows Fig. 2, starting when the Project Manager identifies a to-do activity and want it to be developed. He makes an announcement, sending a message with the performative CFP (Call For Proposals) and the receivers (available developers) may answer with a PROPOSE message containing the cost for the work. After evaluating the proposals, the winner developer is notified with a ACCEPT-PROPOSAL message and the auction losers receive a REJECT-PROPOSAL. Here we see clearly the establishment of a contract between the PM and the winner of the auction and it starts working in order to get the activity done. Finally, when the task is done developers send the result of the job with the activity concluded with an INFORM message to the Project Manager. When all activities are finished, the PM sends an INFORM with the completed set of activities and waits for receiving more. The Project Owner, in turn, answers with another INFORM containing a new set of activities to re-start the development cycle.

Finally, we have to discuss the strategies for deciding the winners in the auction mechanism. The final cost of a project summing up the rework cases could exceed the total budget if the Project Manager does not take this into account. Thus, the reasoning process of deciding which proposal is the best at a specific time could follow many strategies, from choosing the cheapest one to personal choices based on previous experience. Many development teams consider only the cost of development as decision criteria. This is not a good decision because most of the time, cheap bids have higher probability of rework and reassigned activities could even imply higher costs in the future. We call this the simple case. In the other hand, more complex strategies could be developed, considering many other factors to compose a decision. In this case, a more advanced technique is required. Since we are dealing with agents, we decided to improve the criteria used by the PM and give it some learning mechanism in order to observe past results and try to predict the best opportunity to follow. We have opted to use the Q-Learning [20], a reinforcement learning algorithm that tries to find and select an optimal policy function for choosing specific actions given the current state. This function represents the rules that the agent

will follow when giving some state-action pair. After constructing the function, the optimal path is given by selecting the action with the highest value in each state. We take advantage from Q-learning strength since it does not need a previous model of the environment.

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \Bigg[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learnt value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \Bigg] \tag{1}$$

The algorithm works following Eq. (1), where the function calculates the Quality (Q) of a state-action combination. Our Q-learning states are defined as the current states of the project, regarding all the process of selecting developers and the result of their work impacting project cost and rework. The reward of choosing a developer is given by the difference between the budget and project's total cost at that given time, weighted by the probability of rework, seen in Eq. (2).

$$R(t) = (Budget(t) - FinalCost(t)) \times (1 - rework) \tag{2}$$

The learning rate α determines to what extent the newly acquired information will override the old information. A factor of 0 would make the agent not learn anything, while a factor of 1 would make it consider only the most recent information. On the other hand, the discount factor γ determines the importance of future rewards. A factor of 0 will make the agent or short-sighted, only considering current results, while a factor near 1 will make it strive for a long-term high reward.

We have conducted a set of experiments to compare the simple case without optimisation and others trying to find the optimal values of these parameters. Varying α and γ from 0.1 to 1.0 we analysed the influence on the number of members and volunteers chosen and on costs. Due to space limitations we only reference here the achieved optimal values of $\alpha = 0.3$ and $\gamma = 0.7$. Our conclusions were that with these values, the number of auctions was closer to the number of activities and mean numbers of selected members and volunteers were approximately equal. On the other hand, this configuration leaded us to minimal rework and final project costs. In all our experiments reported in Sect. 5 we use this optimal parameter values for α and γ.

4.1 Rework Visualisation

MAESTROS we have the opportunity to inspect all living agents in the environment and see their properties, together with the complete control of the communication flow between agents, given by JADE platform. In contrast, JADE is not a simulation tool and facing the lack of graphical visualisation is a limitation. We have developed then a graphical interface for analysing projects characteristics during simulations

Fig. 3 MAESTROS Status chart (*left*) and Costs chart (*right*)

using JFreeCharts. This module allows us to visualise project's states during development. In Fig. 3 we see two important charts presented on MAESTROS visualisation module: Activity Status Chart and the Costs Chart. The Activity Status Chart shows how activities states change over the time and how rework manifestations, shown when the number of to-do activities increases, indicating that some activities re-entered on the stack for development, indicating clear rework cases. Meanwhile, the Costs Chart shows how costs evolve in terms of budget, estimated cost of the project, rework and final cost. The estimated cost is an important metric since it can be defined as the sum of all activities estimated costs and interpreted as a measure of the optimal minimum cost of the project.

5 Simulation Experiments

We wanted to test MAESTROS capabilities on simulating the OSS environment and verify our model. For that, we have designed two experimental scenarios: (1) Task assignment performance; (2) Impact of project characteristics (activities, budget and developers).

First Scenario—Analysing task assignment
We have designed a default workload that consists on simulating the development of a simple set of activities many times. In a project consisting on 5 activities and 10 available developers (5 members and 5 volunteers) on the environment, the project manager should lead the development trying to keep the final cost closer to the estimated cost and trying not to overpass the budget. The budget is set as 200 % of the estimated cost and to be more realistic, activities have a variable estimated cost (simulating different degrees of difficulty on the tasks) which is randomly set between predetermined values of 5 and 15. Members are allowed to bid between 8 and 15 with 0.1 rework probability (lower) and volunteers bid between 1 and 15 with a 0.3 rework probability (higher), according to literature.

The experiment consists on running the simulation 100 consecutive times and check if the learning mechanism really helps to reduce project's final cost while trying to get lower rework cost. Once each run is independent, the estimated cost,

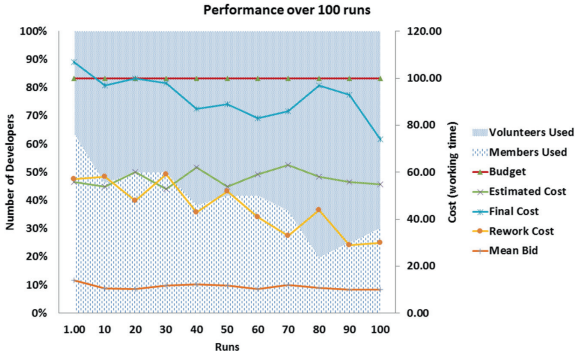

Fig. 4 Performance over 100 runs

final cost and number of auctions may be different. In each run, MAESTROS assigns tasks to developers and records all costs.

The results of the simulation are condensed in Fig. 4. We have overlapped the graphical information of choosing members or volunteers with the project cost results. The shadowed area correspond to the percentage of developers that won the auctions through project development. It seems that at first Project Managers actions were more erratic, leading to a higher final cost that extrapolated both budget and estimated cost. We could see clearly that through the runs the final cost seemed to decrease under the budget and get closer to the optimal cost.

We have no doubt that MAESTROS was able to reduce the rework on this experiment. Comparing the first and the last runs on Table 1 we see the reduction on costs: final cost was reduced in 31 % and rework cost in 47 %. The winner bid also decreased a 29 %. Comparatively, the number of winner members was reduced in 57 % facing an increasing of 75 % on the number of volunteer winners. The number of auctions seemed to be more stable, suffering a small increase of 9 %.

Second Scenario—Different project characteristics

In the second scenario our experiments were focused on verifying how the system performed with different projects characteristics. We have setup and experimented different workloads. In each experiment, focusing in just one property we have varied

Table 1 Performance Improvements

	1st run	100th run	Improv. (%)
Final Cost	107.00	74.00	−31
Rework Cost	57.00	30.00	−47
Mean Bid	14.00	10.00	−29
Members	7.00	3.00	−57
Volunteers	4.00	7.00	75
#Auctions	11.00	12.00	9

its values to check the outcomes after 100 runs. The analysed properties were: **(1) Number of Activities:** Varying from projects with 1 simple activity and ending with 100 activities; **(2) Budget:** Fixing lower project budgets starting from 200 % and ending with only the estimated cost; We have followed the same characteristics described on the first scenario for the costs range of activities and bids, as also used the same probabilities of rework.

The results from running the system 100 times with different number of activities show that this is an important factor that influences projects costs. As seen in Fig. 5, MAESTROS managed to get the final cost under the budget and rework rate very stable, despite the increasing project final cost. When analysing the budget we wanted to see if the learning mechanism in MAESTROS could give us better final costs, trying not to exceed it and looking for the optimal value (estimated cost). In Fig. 6 we see that a broader margin to rework has lead to higher costs, almost 250 % over the estimated, but reducing the budget leaded to decreasing rework and the final cost. Results indicate that at best, MAESTROS achieved a rework cost of approximately 11 % of the final cost.

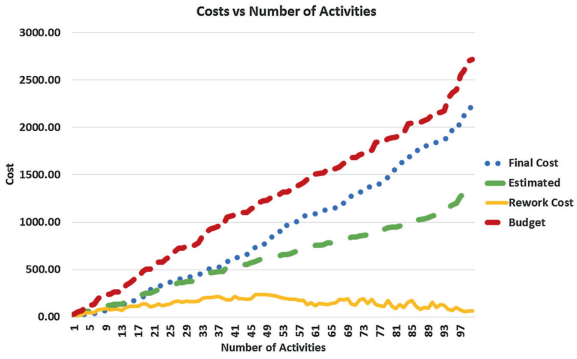

Fig. 5 Varying activities number

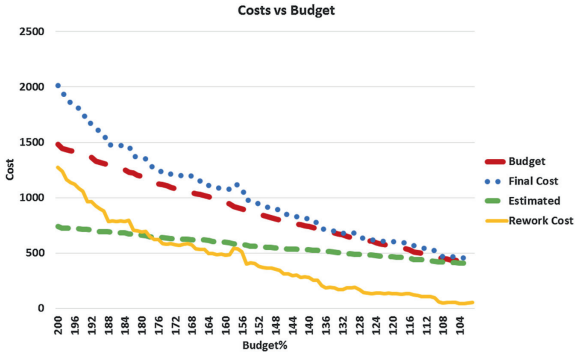

Fig. 6 Varying the budget

6 Conclusions and Future Work

MAESTROS is the first multi-agent approach that study how rework affects Open Source Software projects. This mechanism for activity assignment proved to reduce rework and final projects costs and could be used as an auxiliary tool in real on-line OSS platforms. Our experiments show that multi-agent systems are a good tool to model and simulate a software engineering environment, namely the Open Source Software. MAESTROS has its importance related to the lack of tools and simulations about rework on software development and could help to improve management decisions in this kind of management.

One of MAESTROS most important contributions is that it could help to reduce rework in real projects if the system parameters are fine-tuned with real projects characteristics. MAESTROS could reduce the final cost to a near optimal value (estimated cost of the project) and in comparison to the literature, where the value of 20 % is accepted as a usual rework cost, in MAESTROS we have managed to achieve a rework cost on the order or 11 % of the final cost in average.

We envisage to expand MAESTROS capabilities by providing access to real world projects data from known OSS development environments. We aim to create a good database about open source projects and development teams characteristics and costs. Other future works may include the expansion to model developers work, namely code generation and documentation of activities.

Acknowledgments This work has been funded through a IBRASIL Grant. IBRASIL is a Full Doctorate programme selected under Erasmus Mundus, Action 2 STRAND 1, Lot 16 and coordinated by University of Lille.

References

1. Gaff, B.M., Ploussios, G.J.: Open source software. Computer **45**(6), 9–11 (2012)
2. Software risk management. Springer, Berlin (1989)
3. Chua, B.B., Verner, J.: Examining requirements change rework effort: a study. arXiv preprint arXiv:1007.5126 (2010)
4. Sen, R., Singh, S.S., Borle, S.: Open source software success: measures and analysis. Decis. Support Syst. **52**(2), 364–372 (2012)
5. Raymond, E.: The cathedral and the bazaar. Knowl. Technol. Policy **12**(3), 23–49 (1999)
6. Raja, U., Tretter, M.J.: Defining and evaluating a measure of open source project survivability. IEEE Trans. Softw. Eng. **38**(1), 163–174 (2012)
7. Zahra, S., Nazir, A., Khalid, A., Raana, A., Majeed, M.N.: Performing inquisitive study of pm traits desirable for project progress. Int. J. Mod. Educ. Comput. Sci. (IJMECS) **6**(2) 41 (2014)
8. Rúbio, T.R., Gulo, C.A.: Characterizing developers rework on github open source projects. In: Proceedings of the 10th Doctoral Symposium in Informatics Engineering, FEUP Edicoes - Faculty of Engineering, University of Porto
9. 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. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 222–231. ACM (2014)

10. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of teams for free/libre open source software development. Inf. Softw. Technol. **49**(6), 564–575 (2007)
11. Larman, C.: Agile and iterative development: a manager's guide. Addison-Wesley Professional, Boston (2004)
12. Warsta, J., Abrahamsson, P.: Is open source software development essentially an agile method. In: Proceedings of the 3rd Workshop on Open Source Software Engineering, Citeseer, pp. 143–147 (2003)
13. Wooldridge, M., Jennings, N.R.: Intelligent agents: theory and practice. Knowl. Eng. Rev. **10**(02), 115–152 (1995)
14. Feller, J., Fitzgerald, B.: A framework analysis of the open source software development paradigm. In: Proceedings of the twenty first international conference on Information systems, Association for Information Systems, pp. 58–69 (2000)
15. Koch, S.: Free/open source software development. IGI Global, Hershey (2005)
16. Madey, G., Freeh, V., Tynan, R.: Agent-based modeling of open source using swarm. In: Proceedings of the AMCIS 2002, p. 201 (2002)
17. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: Jadea java agent development framework. In: Multi-Agent Programming, pp. 125–147. Springer, New York (2005)
18. Fipa, A.: Fipa acl message structure specification. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00061/SC00061G.html (2002). Accessed 30 Jun 2004
19. Smith, R.: The contract net protocol: Highlevel communication and control in a distributed problem solver. IEEE Trans. Comput. C **29**, 12 (1980)
20. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)