Agent-based Simulation of Open Source Software Evolution

Conference or Workshop Item

Users and Developers: an Agent-Based Simulation of Open Source Software Evolution

Neil Smith[1], Andrea Capiluppi[2], and Juan Fernández-Ramil[1]

[1] Centre for Research in Computing, The Open University,
Milton Keynes, MK7 6AA, UK
{N.Smith,J.F.Ramil}@open.ac.uk
WWW home page: http://mcs.open.ac.uk/{ns938/,jfr46/}
[2] University of Lincoln, Brayford Pool, Lincoln, LN6 7TS, UK
acapiluppi@lincoln.ac.uk
WWW home page: http://hemswell.lincoln.ac.uk/~acapiluppi/

Abstract. We present an agent-based simulation model of open source software (OSS). To our knowledge, this is the first model of OSS evolution that includes four significant factors: productivity limited by the complexity of software modules, the software's fitness for purpose, the motivation of developers, and the role of users in defining requirements. The model was evaluated by comparing the simulated results against four measures of software evolution (system size, proportion of highly complex modules, level of complexity control work, and distribution of changes) for four large OSS systems. The simulated results resembled all the observed data, including alternating periods of growth and stagnation. The fidelity of the model suggests that the factors included here have significant effects on the evolution of OSS systems.
Keywords: simulation models, software process, open source software, software evolution.

1 Introduction

Computing is a rapidly evolving discipline and there is a need to understand the evolutionary processes that prevail in new forms of software development, such as open source software (OSS) systems. Evolution in proprietary systems is becoming understood [1], but many OSS systems do not evolve in the same way [2, 3]. This suggests that existing theories of software evolution are partial accounts of OSS evolution. Extending these theories can have a practical output by informing good practice, leading to the more efficient prodution of better software. This paper reports our attempts to use theories of software evolution to replicate and explain empirical observations of a set of OSS systems.

OSS evolution involves a community of individuals providing their work mainly on a voluntary basis and without a strong centralised leadership [4, 5]. This invalidates one of the assumptions of many simulation models: the existence of a centralised management which reacts to the state of the software system by altering the pattern of work performed [6]. This emphasis on individuals suggests that an agent-based model of the OSS evolution process is appropriate [7, 8].

We propose that each module within an OSS system is monolithic and will behave as such [1, 9]. However, the modular architecture will restrict the impact of growth stagnation to small parts of the system, where it will not have a significant global effect. To investigate this hypothesis, we have developed an agent-based model of OSS development. To our knowledge, the model presented here is the first model of open source evolution that includes four significant factors: the complexity of the software modules as a limiting factor in productivity; the fitness of the software to the requirements; the motivation of developers; and the role of users in defining requirements. We believe, based on our experiments, that these are important factors that need to be included in OSS evolution theories.

2 Agent-based Simulation Model

Our motivation for developing this model lies in our understanding of the actions of individual OSS developers [10, 4]. OSS development is decentralised and non-coercive: generally, developers choose to become involved in an OSS project and choose which aspects of the project to work on. This focus on individual developers and specific software components suggests that such objects should be the primitive elements of our model, with the evolution of the OSS system being an emergent property of the interactions of those primitive elements.

We used the NetLogo [11] multi-agent simulation tool. In this tool, agents move around a virtual world (a grid of "patches"), interacting with it and with other agents. Each agent and patch has its own state and procedures. Simulation proceeds by each agent and patch performing its behaviour independently, often by following stochastic functions influenced by the agent's state and local environment. Agents perform their own actions asynchronously; there is no centralised co-ordination of the agents' actions.

In our model, patches represent *modules* of software source code and different types of agents represent *developers*, unfulfilled *requirements*, and *users*.

A *module* is a single modular part of a software system. Modules that are near each other are functionally related. Each module records both its fitness for purpose and its complexity. The complexity of a module acts as an inhibitor to future changes to that module. To model the changes in external requirements that drive software evolution, patches have a stochastic process for decreasing their fitness over time. Finally, modules have a chance to capture the attention of a developer passing through cyberspace and so create a new developer agent in the model; this only happens if the module is interesting (i.e. its fitness is below the developer's 'boredom threshold'; see below).

Users are responsible for adding new requirements to the system. Users walk randomly around the system space and, when they meet a code patch or an existing requirement, they create a new *requirement* that extends the functionality in this area. Newly created requirements attract new users, which reflects the tendency of users to suggest modifications to existing requirements.

Fig. 1. An example of simulated OSS development. The squares are code patches. The black circles are unfulfilled requirements. Note how the vertical "stem" in the top left is loosely connected to the rest of the system.

Developers walk randomly around the software system, changing code as they go. Agents have four behaviours, depending on their location. If a developer is on an unfulfilled requirement, it creates a *new module* that fulfils that requirement, with a certain (low) fitness and complexity. If a developer is on a module with high complexity and high fitness, it may attempt to *refactor* that module. Refactoring leaves the module's fitness unchanged, but reduces its complexity by a random amount. If the developer chooses not to refactor a module, it will attempt to *develop* the module: this increases the module's fitness and complexity by a random amount. However, if the module is complex, the agent may not be able to improve the module, in which case the module is left unchanged. Finally, developers have a *boredom threshold*. If the fitness of the module they are on is above this threshold, there is a chance that the developer will find the project boring and leave. Developers may also leave if they wander onto a patch and have no module or requirement to work on.

Simulation starts with a single module and a single user. These spawn new requirements and attract the attention of developers. The developers create modules to fulfil the requirements. As the project grows, more developers and users are attracted and more requirements are identified. The model's source code is available from `http://mcs.open.ac.uk/ns938/simulation/`.

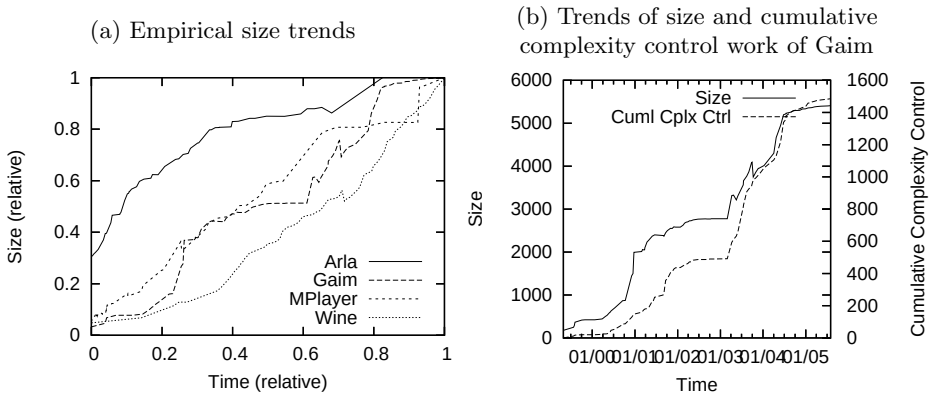
3 Empirical Data

To validate the model, we compared the simulated output to empirically observed behaviour. The empirical data was derived from data in OSS repositories. Previous research has shown that data such as change-log records, program headers and configuration management offer a suitable source of data for the study of

Table 1. Data sources used

Software System Studied (URL of Code Repository)	Change Log?	CVS?	Number of Releases Considered
Arla (www.stacken.kth.se/projekt/arla)	Yes	Yes	70
Gaim (http://gaim.sourceforge.net)	N/A	Partial	100
MPlayer (www.mplayerhq.hu)	N/A	Partial	81
Wine (www.winehq.com)	Yes	Partial	90

Fig. 2. Empirical trends



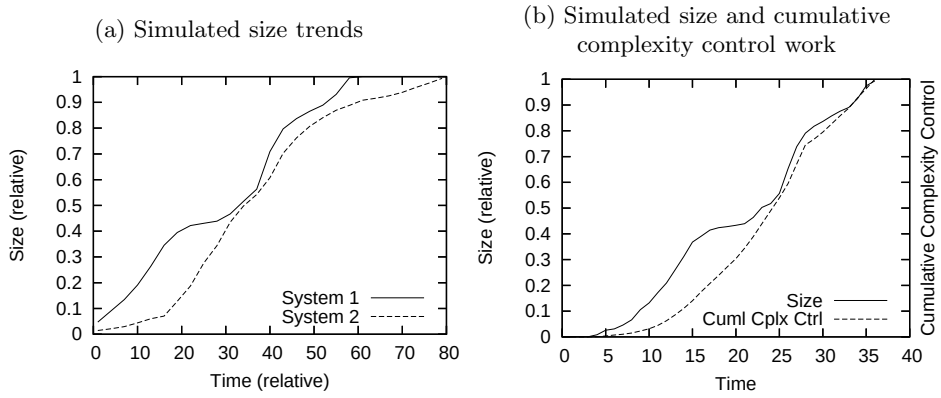
software evolution [12–14]. For this study, we selected four OSS systems which we have examined in previous studies [13, 14]. Table 1 indicates the data sources we used to extract the empirical data used in this research. We extracted several attributes for each software system, taking measurements over releases.

Size was evaluated using number of source functions (as a surrogate for the systems' growth). Figure 2(a) shows the size trends for all the systems, using relative sizes and times. Only Wine has a smoothly increasing trend; the other systems had at least one period of reduced growth (i.e. stagnation).

Complexity was measured at the level of functions. We used the McCabe cyclomatic number [15] as a measure of complexity and the accepted threshold value of 15 to distinguish highly complex functions [16]. In all the analysed systems, the highly complex functions never make up more than 10% of the overall system.

We measured *complexity control work* by comparing every function between two consecutive releases and counting how many of them reduced in complexity. There is a high correlation between the trend of the size growth and the cumulative amount of complexity control work: figure 2(b) shows this for Gaim.

Fig. 3. Simulated trends



A function is *touched* when it is added, deleted, or modified. A small subset of elements is touched a large number of times by developers, whilst most of the elements receive few touches. The skewness of these distributions ranges from 2.73 in Wine to 4.55 in Arla.

4 Results and Validation

We used the empirical data described above to calibrate and evaluate our model. We did this by exploring the parameter space of the models, looking at the generated output, and comparing it to the empirical evidence from the four OSS systems. Throughout most of the parameter space, the model generated results that were very similar to the empirical results.

The model was most sensitive to the value of the boredom threshold parameter, which controls when new developers join and leave the project. If the boredom threshold of developers was high, high-fitness modules attracted developers rather than forcing them to leave. The number of developers grew rapidly and soon swamped the development environment. In contrast, if the boredom threshold was low, the evolution of the first few modules resulted in a system that attracted no new developers; the original developers soon left and the project became moribund.

The behaviour of the users was important to produce both discontinuous and smooth growth patterns (figure 3(a)). Without the users clustering around new requirements, the requirements were spread evenly around the system and only smooth growth patterns were produced [17]. Even with the clustering, some simulations produced smooth growth.

In the simulations, the proportion of complex functions remained at a constant and low level as long as new modules have an initial complexity below the reporting threshold, similar to the empirical results. This behaviour was not seen if refactoring was ineffectual or not attempted. Moving to complexity control work, figure 3(b) shows that the simulation reproduces the empirical pattern of increasing complexity control work that eventually follows the growth trend.

Finally the simulation is able to partially reproduce the long-tailed distribution of touches, though the skew value is typically only 0.8–0.9.

The closeness of the simulated results to the empirical data indicate that our model reflects many of the processes that occur in OSS evolution.

5 Related Work

The OSS domain was originally studied using quantitative metric data extracted from OSS systems [10, 13, 14]. Godfrey & Tu [2] highlighted differences between the evolution of Linux and previously studied systems, particularly its apparently super-linear growth. Our model provides a possible explanation for such a super-linear growth: access to an effectively unlimited pool of developers and complexity which constrains productivity to the module level only.

Antoniades *et al.*'s [18] simulation of OSS processes has reproduced empirically observed patterns of growth and developer numbers. Robles *et al.* [19] propose a biologically-inspired simulation, where developers learn from other developers only through observing changes in the source code. Their research shares our focus on product characteristics (e.g. size and complexity) and on evolution. However, to our knowledge, the model presented here is the first model of open source evolution that includes the complexity of the software modules as a limiting factor in productivity, the fitness of the software to the requirements, and the motivation of developers.

6 Further Work

Our further work will initially focus on two areas. First is the surprising need to include the behaviour of the users in creating new requirements. As far as we know, the role of users in the evolution of OSS systems has not been deeply explored. We will examine the empirical data to try to identify how and when users generate new requirements and how they are dealt with by developers. Second is inability of the model to produce touch distributions that were as skewed as the empirical data. This may be due to the undiscriminating behaviour of the developer agents in modifying existing code patches and the initial values of a module's fitness and complexity: it appears that many modules in real systems are rarely touched because their initial implementation is adequate and not subject to change.

The other aspect of developer behaviour that we will soon add to the model is to take account of developers' experience in controlling and approving changes made to the system. In many OSS systems, there is a core of highly experienced

super-developers that have a great influence on the evolution of the system [20]. We anticipate that including such super-developers in the model will have a significant effect on the simulation.

7 Conclusions

This paper presented an agent-based simulation model of OSS evolution. Our model, while simple, incorporates many of the features that may explain some of the differences between OSS and proprietary development [2, 3]. We found that the model was able to replicate the observed patterns in all four of the areas examined (size, complexity, complexity control, distribution of changes) in the four systems studied. The model presented here appears to provide an explanation for the unbounded growth trends observed in some OSS software [2, 3]. This is an important contribution. We included four novel factors in our model: the complexity of software modules as a limiting factor in productivity, the fitness of the software to its requirements, the motivation of developers, and the role of users in incrementally defining requirements. As discussed in section 4, all four of these factors are required for the model to produce plausible results.

In conclusion, we have shown that an agent-based model of OSS evolution can faithfully produce the empirical behaviour of OSS systems, but only by including a number of factors that are not immediately obvious. This suggests that studies into the factors driving software evolution need to look beyond just the behaviour of developers.

Acknowledgements

Andrea Capiluppi acknowledges the Faculty of Maths and Computing, The Open University, and in particular to Drs Bashar Nuseibeh and Uwe Grimm, for financial support that made this work possible. Juan Fernández-Ramil gratefully acknowledges the UK EPSRC for funding under grant GR/590782/01 (2004–5).

References

1. Lehman, M.M., Fernández-Ramil, J.: Software Evolution. In: Software Evaluation and Feedback — Theory and Practice. Wiley (2006)
2. Godfrey, M., Tu, Q.: Growth, evolution and structural change in open source software. In: Proceedings of the 4th International Workshop on the Principles of Software Evolution, Vienna, Austria (2001)
3. Herraiz, I., Robles, G., Gonzalez-Barahona, J.M., Capiluppi, A., Fernández-Ramil, J.: Comparison between SLOCs and number of files as size metrics for software evolution analysis. In: Proceedings, 10th European Conference on Software Maintenance and Reengineering. (2006)
4. Raymond, E.S.: The Cathedral and the Bazaar. O'Reilly Media, Inc. (2001)
5. Scacchi, W.: Understanding Open Source Software Evolution. In: Software Evolution and Feedback, Theory and Practice. Wiley, NY (2006)

6. Smith, N., Capiluppi, A., Fernández-Ramil, J.: A study of open source software evolution data using qualitative simulation. Software Process Improvement and Practice **10** (2005) 287–300
7. Madey, G., Freeh, V.W., Tynan, R.O.: Agent-based modeling of open source using SWARM. In: Proceedings of the Americas Conference on Information Systems (AMCIS 2002), Dallas, USA (2002)
8. Dalle, J.M., David, P.A.: imCode: Agent-based simulation modelling of open-source software development. Technical report, MIT (2004)
9. Brooks, F.: The Mythical Man-Month: Essays on Software Engineering. 20th anniversary edn. Addison-Wesley (1995)
10. Mockus, A., Fielding, R.T., Herbsleb, J.: Two case studies of open source software development: Apache and mozilla. ACM Transactions Software Engineering and Methodology **11**(3) (2002) 309–346
11. NetLogo: http://ccl.northwestern.edu/netlogo/ (2005)
12. Capiluppi, A.: Models for the evolution of OS projects. In: Proceedings, ICSM 2003, Amsterdam (2003) 65–74
13. Capiluppi, A., Morisio, M., Fernández-Ramil, J.: The evolution of source folder structure in actively evolved open source systems. In: Proceedings of the 10th International Symposium on Software Metrics, Chicago, USA (2004) 2–13
14. Capiluppi, A., Morisio, M., Fernández-Ramil, J.: Structural evolution of an open source system: A case study. In: Proceedings of the 12th International Workshop on Program Comprehension (IWPC), Bari, Italy (2004) 172–182
15. McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering **2** (1976) 308–320
16. McCabe, T.J., Butler, C.W.: Design complexity measurement and testing. Communications of the ACM **32**(12) (1989) 1415–1425
17. Smith, N., Capiluppi, A., Fernández-Ramil, J.: Agent-based simulation of open source evolution. Software Process Improvement and Practice (to appear)
18. Antoniades, P., Samoladas, I., Stamelos, I., Bleris, G.L.: Dynamical simulation models of the Open Source Development process. In: Free/Open Source Software Development. Idea Group, Inc. (2005)
19. Robles, G., Merelo, J.J., Gonzalez-Barahona, J.M.: Self-organized development in libre software: a model based on the stigmergy concept. In: ProSim 2005, St. Louis, USA (2005)
20. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the apache server. In: Proc. ICSE 22, Limerick, Ireland (2000) pp. 263 – 272