Agent-based Simulation of Open Source Evolution

Neil Smith, Andrea Capiluppi, Juan Fernández Ramil
Computing Department
The Open University
Walton Hall, Milton Keynes MK7 6AA, U.K.
{n.smith, a.capiluppi. j.f.ramil}@open.ac.uk

Abstract

We present an agent-based simulation model developed to study how size, complexity and effort relate to each other in the development of open source software (OSS). In the model, many developer agents generate, extend, and refactor code modules independently and in parallel. This accords with empirical observations of OSS development. To our knowledge, this is the first model of OSS evolution that includes the complexity of software modules as a limiting factor in productivity, the fitness of the software to its requirements, and the motivation of developers.

Validation of the model was done 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 the observed data, except for system size: three of the OSS systems showed alternating patterns of superlinear and sublinear growth while the simulations only produced superlinear growth. However, the fidelity of the model for the other measures suggests that developer motivation and the limiting effect of complexity on productivity have a significant effect on the development of OSS systems and should be considered in any models of OSS development.

Keywords: simulation models, software process, open source software, software evolution, productivity, metrics, Agent-based simulation.

1. Introduction

The application of *simulation models* to software processes includes the support of decisions about resources and the impact of process improvements, such as the introduction of inspections. Simulation modelling can also help in evaluating possible explanations for empirical observations. The vast majority of real world software applications are *evolved* from existing versions, not created from scratch, and a large amount of effort go into evolution, not into initial development (Sommerville 2001). There is now a body of knowledge about software evolution (e.g. Lehman & Belady 1985, Rajlich & Bennett 2000, Aoyama 2002, Madhavji *et al.* 2006) based on observations and on a variety of software process models (e.g. Lehman *et al.* 2002, Smith *et al.* 2005). In spite of the advances over the years, the management of software evolution is one of the most challenging software engineering problems. In order to make progress in this area, we are interested in understanding the drivers of software evolution, as a basis to the generation of *theories* of software evolution.

This is an important topic since software engineering lacks an empirically validated theory (Lehman 2000a). Even though this could be seen as a topic of interest to academics only, it can have a practical and useful output in terms of guidance and justification for good practice (Lehman 2000b). Guidance to good practice is one of the roles of theory in other engineering disciplines. For example, one can use electromagnetic field theory to reduce electromagnetic interference between nearby cables. In a similar way, one could consider Brooks's law (Brooks 1995) that "adding people to a late project make it later" to derive guidelines for managing a late project. Similarly, Lehman's first law of software evolution, "Continual Change" (Lehman 1974), can justify the introduction of configuration and change management in the software process (Lehman 2000b). Due to the human involvement at all levels in the software process we expect that the theories related to the software process will be more qualitative than the quantitative theories typical of physical systems. If software processes are subject to general influences that can be modelled and captured in theories, then process simulation modelling will play a role in evaluating the empirical support for them. The investigation of such theories should include not only the study of empirical data in search of

patterns, but also the investigation of ways of abstracting and representing any behavioural regularities found, and of the limits of such theories. For example, we would like to know whether theories are able to explain and predict break points in otherwise apparently regular evolution.

This paper reports our attempts to replicate empirical observations of size, complexity, refactoring work and distribution of changes of a set of *open source software* (OSS) systems. The free availability of empirical data makes OSS an attractive topic for investigation. OSS development projects are very modular: multiple developers work in parallel on different parts of a software system with less influence from a formal plan than in proprietary software development. This suggests that an *agent based* model of the software development process would be appropriate. The use of such a model was proposed some 10 years ago (Lehman & Stenning 1996). However, to our knowledge, the model presented here is the first model of open source evolution that includes three significant factors: 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. This is an advance over the work of other researchers (Madey *et al.* 2002, Robles *et al.*. 2005). These OSS simulations have dealt with mainly two dimensions (system size and number of developers) and how an increased number of developers produces more output (lines of code and source files).

The structure of this paper is as follows: Section 2 provides the motivation for this research. Section 3 introduces our agents-based model for simulating an OSS environment, the assumptions on which it is based, and the hypotheses of our research. Section 4 describes the empirical data and the system from which it is derived. Section 5 compares the simulation output with the empirical data. Section 6 describes related work in simulation of OSS software evolution. Section 7 concludes the paper and indicates topics for further work.

2. Motivation and Background

Many simulations of software development are based on the traditional approach to modelling complex systems. The system's behaviour is abstracted into a set of differential equations which define the structure of the causal links between the state variables in the model (Iwasaki & Simon 1986); this describes the mechanisms that are believed to operate in the referent system. The set of equations is solved to produce a behaviour, which is compared to the empirical data. This approach has produced results in the study of traditional software development (e.g. Abdel-Hamid & Madnick 1991) and has produced predictions such as the diminishing growth rate implied by Lehman's second law of increasing complexity (Lehman 1974) and observed in a number of proprietary software systems (Turski 1996, FEAST 2001, Turski 2002). In our previous work we explored the use of qualitative simulation to bridge the gap between high level (qualitative) theories and empirical data (Ramil & Smith 2002, Smith *et al.* 2005).

Computing is a rapidly evolving disciplineand there is a need to compare the theories to new emergent forms of software development, such as the OSS domain. This presents some challenges. Many OSS systems do not show the types of evolutionary behaviour seen in the evolution of traditional software (Godfrey & Tu 2002, Herraiz 2005). About one-third of OSS systems grow at super-linear rates (seemingly contradicting Lehman's second and fifth laws) (Herraiz 2005). In OSS, the inclusion of more contributors leads to more growth (also contradicting both Lehman's second and fifth laws and Brooks's law).

This different behaviour may be due to the different architectural structure of OSS development: our hypothesis is that OSS development behaves differently because OSS systems are more modular than proprietary software. We believe that the distributed nature of OSS applies not only to the software itself but also at the community evolving it (also following Brooks's observation that the architecture of a system reflects the structure of the organisation evolving it (Brooks 1995)). OSS evolution involves a community of individuals providing their work mainly on a voluntary basis and without a strong centralised leadership (Raymond 2001, Scacchi 2006). This invalidates one of the assumptions of our previous simulation models: the existence of a centralised management control which will react against excessive complexity by assigning developers to complexity control work (Smith *et al.* 2005).

We propose that, while each module within an OSS system may be monolithic, and will behave in the manner described by Lehman, Belady and Brooks, the overall modular and decentralised

architecture of the software and of the community evolving it will restrict the impact of software growth stagnation to small parts of the system where they will not have a significant effect on the evolution of the whole. To investigate this hypothesis, we have developed an agent-based model (Rocha 2003) that captures the decentralised and modular nature of OSS development. The next section describes the details of the model and section 5 shows the simulation results generated by it.

3. Agent-based Simulation Model

Our simulation model is based on the Lehman's laws of software evolution (Lehman 1974, Lehman & Belady 1985, Lehman 2000b), our own experience observing OSS development, as well as on the models developed by other researchers (Antoniades *et al.* 2005, Robles *et al.* 2005, Madey *et al.* 2002). However, to our knowledge, the model presented here is the first agent-based 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. We believe that these are important factors that need to be studied.

Our motivation for developing this model lies in our understanding of the actions of individual OSS developers (Mockus *et al.* 2002, Raymond 2001). OSS development is inherently decentralised and non-coercive: developers choose to become involved in an OSS project and choose what aspects of the project to work on. They generally have pride in their work and take pains to ensure that it is easily maintainable.

We used the NetLogo (2005) multi-agent simulation tool to develop our model. We selected NetLogo primarily because it is freely available on the web and well documented and supported. In this tool, agents move around a virtual world, interacting with it and with other agents. There is no centralised control or co-ordination of the agents' actions. Agents are responsible to maintaining their own state. The NetLogo virtual world consists of a grid of "patches", each of which can have a state. Generally, agents have only local knowledge about their surroundings. Both agents and patches are active agents in the simulation, performing actions and asking other agents to perform other actions. Simulation proceeds by each agent and patch repeating its behaviour independently, often by following stochastic functions influenced by the agent's state and local environment. Agents perform their own actions asynchronously and as rapidly as they can. In an agent-based simulation, the overall behaviour of the system is an emergent property of the individual, independent interactions of the agents. This approach differs from the traditional modelling approach where the state of the system is captured in a single set of global state variables, such as stocks and flows.

In our model, patches represent *modules* of software source code and agents represent both *developers* and unfulfilled *requirements*. Figure 1 shows a class diagram that illustrates the main concepts of developers interacting in the system by creating, modifying, and refactoring modules. The level of abstraction of the model is not determined a priori: a module could represent a single procedure, a file, a library, or some other modular part of a software system. Each module records both its fitness for purpose (gauged against a set of external requirements) and its complexity. The complexity of a module acts as an inhibitor to changes to that module. To model the changes in external requirements and invalidation of assumptions that are a driving force behind software evolution, patches have a stochastic process for modelling their decrease in fitness. Modules also have a random chance to spawn a new requirement in a neighbouring, empty patch. 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 is the module is interesting (i.e. its fitness is below the developer boredom threshold; see below). In the model, each patch repeatedly exhibits these behaviours, checking if it should generate a new requirement, decrease in fitness, or capture the attention of a new developer agent. These behaviours are shown in Figure 2.

<<Figure 1 approx here>>

In the model, software developers are represented by agents. These agents walk randomly around the software system, changing the code as they go. Agents have four behaviours, depending on their location. As with patches, each developer repeatedly and asynchronously cycles through these behaviours, following the pseudocode in Figure 2.

1. If a developer is on an unfulfilled requirement, it creates a *new module* that fulfils that requirement, with a certain (low) fitness and complexity.
2. 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.
3. If the developer is on a module that it is not refactoring, it will attempt to *develop* the module; this increases both the module's fitness and complexity by a random amount. It will also slightly reduce the fitness of adjacent modules, due to coupling between them. However, if the module is complex, the agent may not be able to improve the module, in which case the module is left unchanged.
4. Finally, developers have a motivation factor (we call this *boredom threshold* in the model). 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 move outside the system.

<<Figure 1 approx here>>

Simulation starts with a single module. This both spawns new requirements and attracts the attention of developers. The developers will create modules to fulfil the requirements and therefore enlarge the project. As the project grows, more developers are attracted and more requirements are identified. The code of the full NetLogo model is freely available from http://mcs.open.ac.uk/ac5468/simulation/.

4. 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 rich source to derive data for the study of software evolution (Capiluppi 2003, Capiluppi *et al.* 2004a,b, Mens *et al.* 2004, Fischer *et al.* 2003, RELEASE 2005). For this study, we selected four OSS systems which we have examined in previous studies (Capiluppi *et al.* 2004a,b). Table 1 indicates the data sources we used to extract the empirical data used in this research.

<<Table 1 approx here>>

We extracted several attributes for each software system, taking measurements over releases for size in number of files, files handled (Lehman & Belady 1985), average complexity (measured using the McCabe index (McCabe 1976)), and the level of complexity control work (measured as the proportion of files/functions which were subject to a decrease in their complexity in two contiguous releases). The data collection was aimed at measuring the systems' size, complexity, amount of anti-regressive work, and distribution of touches: the purpose of this work is to characterise an OSS environment in order to compare it against the simulated results obtained from the model described above.

- **Measurement of size of the system:** size was evaluated using number of source functions (as a surrogate for the systems' growth). When release data was available, the code base and its size at each release were evaluated using our tools. We believe that an approach based on measuring size of the released source code more genuinely displays the overall evolution of the system than measuring the size of the whole repository. Releases contain those parts of code that were chosen to be included in what is normally called a *stable configuration*. We prefer releases for the study of size of code contained onto the whole CVS server, because they represent stable points. The right-most column in Table 1 indicates the number of releases studied. The evaluation of the size achieved is displayed in a joint visualisation in Figure 3a for all the projects: the X axis indicates the time of each release, while the Y axis indicates the size achieved. Measures are relative to the maximum values of both metrics, in order to allow the comparison of systems with largely different values in size achieved. As can be seen, only one case (Wine) exhibited a monotonically increasing trend; in the other systems at least one period of stagnation was observed between two periods of growth. Figure 3b shows one of the cases with a discontinuous pattern of growth (Gaim system).

<<Figure 3a and 3b approx here>>

- **Measurement of complexity:** We looked at complexity at the level of granularity of functions. Within this level we consider the McCabe cyclomatic number as a measure of complexity (McCabe 1976). Next, we define the *highly complex* subpart of the system as the set of highly complex functions. We use the accepted threshold value for the McCabe index of 15 in order to distinguish between less complex functions and more complex ones (McCabe & Butler 1989). One of the proposed systems had already been evaluated (Capiluppi *et al.* 2005) for the purpose of tracking the amount of highly complex functions: a similar approach has been used for the other systems in this case study. This is shown through a boxplot [Box 1978] visualisation, which displays the variation of the dataset (i.e. the percentage of the highly complex elements in a specific system) along the number of its releases. Figure 4a shows that, in all the analysed systems, the highly complex subpart is never larger than 10% of the overall system. Figure 4b shows the overall trend of highly complex functions for two Gaim and Mplayer.

<<Figure 4a and 4b approx here>>

- **Measurement of the level of complexity control work:** An adequate level of complexity control (also termed anti-regressive work Lehman 1974; Lehman & Belady 1985)) is widely recognized as an essential countermeasure to software aging (Parnas 1994), and to sustain the evolution of software (Ratzinger & Gall 2005). We measured the level of complexity control work by comparing every function between two consecutive releases and by counting how many of them experienced a reduction in their cyclomatic complexity. From a quantitative analysis, our results illustrate that there is a correlation between the trend of the size growth and the amount of complexity control work: Figure 5a, 5b, 5c and 5d show the trends for the studied systems.

<<Figure 5a, 5b, 5c and 5d approx here>>

- **Distribution of changes:** We term the number of different releases during which a file has been manipulated, via addition, removal, or modification of code as the *release touches* of that file. A single release touches indicates that the file has never been modified after its creation. The maximum possible number of release touches for a file is the number of releases. The quantitative observation we drew from the chosen case studies, when analysing the distribution of release touches, resembled those already achieved (Capiluppi *et al.* 2005): a small subset of elements (files or functions/methods/classes) is touched a large number of times by developers, whilst most of the elements receive few (if not none) touches (Figure 6). This behaviour can be summarised by visualising, per each system, its release-touches profile, or by giving the skewness of the distribution. The higher the skewness, the more asymmetric the distribution of the release touches in a system.

<<Figure 6 and Table 2 approx here>>

5. Results and validation

We used the empirical data describe above to calibrate and validate 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 case studies. The comparison was done by examining four groups of attributes: growth, complexity complexity control work, and distribution of touches.

We exhaustively explored the parameter space of the model by determining a range of possible values for each of the nine controlling parameters in the model. This resulted in 256 distinct combinations of parameter values. The model was run for each of these parameter combinations and the results logged.

The model seemed most sensitive to the value of the boredom threshold parameter, which controls when new developers join and leave the project (if a module's fitness is below the boredom threshold, it may attract new developers; if the fitness is above the boredom threshold, developers on that module may leave the project). If the boredom threshold of developers was set very high, developers did not leave the project when they encountered high-fitness modules. As these modules continued to attract new developers, the number of developers grew extremely rapidly and soon swamped the development environment. In contrast, if the boredom threshold was very low, the evolution of the first few modules resulted in a system that was sufficiently fit for purpose that no new developers were attracted; the original developers soon left the project and were not replaced.

Neither of these behaviours is at all similar to the empirical data, so the results from these simulation runs were discarded.

The simulation results were also strongly affected by the effectiveness of the refactoring work done by developers. When refactoring was ineffectual or not attempted, the both the average complexity of the system and the proportion of the highly complex subpart grew over time. This in turn led to an unusual behaviour for the average fitness of the system: it peaked in the early stages of development and gradually declined thereafter. This was due to the large number of 'old' modules that had suffered *bitrot* (so they became unfit for purpose) but could not be improved due to their complexity. This also led to the anomalous situation of these modules attracting many functionally ineffectual developers.

The rest of the runs yielded very similar behaviours (up to linear scalings of the various results): the system size grew super-linearly, the proportion of complex modules remained constant and low, the portion of work assigned to complexity control increased over time, and the distribution of touches was very skewed.

- Growth patterns: Figure 3 shows the empirical growth patterns. Figure 7a shows that the simulation model is able to reproduce the continuous growth pattern seen in Wine. The second pattern (discontinuous growth) is not directly reproduced by the model, but it can be generated by varying the controlling parameters during simulation, such as adjusting the chance of new developers appearing (thus changing the number of developers) or changing the threshold that determines when a module adequately meets its requirements (similar to the idea of S-type programs defined by Lehman & Belady (1985)). In the further work section we highlight an extension of the model for capturing the discontinuous trends, based on the fulfilment of the initial requirements.

<<Figure 7a approx here>>

- Complexity patterns: As shown in Figure 4, all of the four systems studied present a similar trend when analysing the percentage of highly complex functions. The simulated data is similar to them (Figure 7b), as long as new modules have an initial complexity below the reporting threshold. Both complex and non-complex modules attract developers who perform some refactoring work and hence keep the average complexity constant. This is very similar to the empirical observations.

<<Figure 7b approx here>>

- Complexity control: in the four systems studied the empirical observations show that the cumulative amount of complexity control (also termed *refactoring*) work, that is, the work aimed at decreasing the complexity of a software system, increases slowly but then closely follows the growth trend. That is, the complexity control rate meets and even exceeds the functional growth rate. As visible in Figure 8a, the simulation model is able to reproduce this pattern. Refactoring work is taking an increase amount of work but this is sustained because in OSS the effort comes from an unbounded pool of developers.

<<Figure 8a approx here>>

- Release Touches: as shown in the empirical observations (Figure 6), the studied software systems show a long-tailed distributions of release touches. The model is able to simulate this aspect. Albeit a qualitative resemblance is obtained, the skewness factor is not very large, as we observed a value of 0.81 for the simulated output. The highly skewed profile is reproduced by our simulation engine, as visible in Figure 8b.

<<Figure 8b approx here>>

6. Related Work

The study of evolutionary trends in software evolution has been the focus of several quantitative simulation models using system dynamics (e.g., Lehman *et al.* 2002). This modelling effort has been inspired by observations of proprietary software. The OSS domain was originally studied using direct visualization of trends based on quantitative metric data extracted from OSS system (Capiluppi *et al.* 2004). A important study by Godfrey & Tu (2001) highlighted differences between the evolution of Linux (a popular OSS operating system) and previously studied systems,

particularly its apparently super-linear growth. Our model provides a possible explanation for such a super-linear growth.

Research efforts specifically involving simulation of some aspects of OSS development and evolution have taken place and this area of research is becoming very popular, driven by the availability of OSS data. Examples include Madey *et al.* (2002), who used the existing SWARM agent-based simulation tools, and Dalle & David (2004), who built their own agent platform, SimCode. A dynamic simulation of OSS processes is also described by Antoniades *et al.* (2005), where empirically observed patterns of size growth and developers joining the project are reproduced by a simulation. Robles *et al.* (2005) propose a biologically-inspired simulation, where developers learn from other developers through observing changes in the source code, rather than explicitly communicating with each other. This type of model has been used to investigate questions related to the amount of effort allocated to OSS projects and whether a significant attraction of new developers can be achieved in the evolution of the project. 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.

7. Conclusions and Further Work

This paper presented an agent-based simulation model of OSS evolution. Our model, while simple, incorporates many of the features that may explain the differences between OSS and proprietary development (Godfrey & Tu 2001, Herraiz 2005). We found that the model was able to replicate the observed patterns in three of the four areas examined (complexity, complexity control, distribution of changes) in the four systems studied. In one area, system growth, the model was only able to replicate the continuous superlinear growth pattern seen in one of the four systems studied. Discontinuous behaviour can be artificially generated by changing parameters during a simulation. However, this is not satisfying as an explanation and more work is needed. Having said that, the model presented here appears to provide an explanation for the unbounded growth trends (Godfrey & Tu 2001, Herraiz 2005) observed in some OSS software. By itself, this is an important contribution.

We included three novel factors in our model: the complexity of software modules as a limiting factor in productivity, the fitness of the software to its requirements, and the motivation of developers. All three of these factors are required for the model to produce plausible results. If the fitness of modules and the interests of developers are misaligned, the model quickly deviates from empirical observations. If the modules created are perfectly fit for purpose, developers have little to do and leave the project, resulting in a moribund project. If the developers never leave the project, no matter how fit for purpose existing modules are, the number of developers grows excessively large. The 'desire' of developers in the model to refactor complex code, which reduces the complexity of that module and hence the system overall, leads to the highly complex subpart of the model to remain at a constant and low level, in accordance with the empirical observations. Experiments where this refactoring behaviour was prevented led to situations where the highly complex subpart grew much faster than the overall size of the system and the average fitness of the system decreased over time. This indicates that refactoring is a significant activity in OSS development that allows the system to remain fit for purpose and able to continuously grow. These results indicate that all of the novel factors introduced in this model are required for faithful simulations of OSS evolution.

The work reported in this paper, particularly the problem in simulating discontinuous trends, has lead to the view that a better model may be developed by modelling the actions of core developers. Observations of some OSS projects indicate that they are led by core developers who perform most of the changes in the software while others make a much smaller number of contributions (Mockus *et al.* 2002). This suggests that we need to include mechanisms in the model which reflect the role of core developers in influencing the evolution of some OSS systems. An alternative way of explaining the discontinuous trends of size growth could be to make the distribution of requirements less even, with clusters of requirements representing new functional areas for the project to

incorporate. This would generate rapid growth while a new area is explored followed by slow growth then those requirements are fulfilled.

8. 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.

9. References

Abdel-Hamid T.K., Madnick S.E., 1991, *Software Project Dynamics – An Integrated Approach*, Prentice Hall, Englewood Cliffs,, NJ: 263 pp

Antoniades P., Samoladas I., Stamelos I., Bleris G.L., 2005, Dynamical simulation models of the Open Source Development process. In Stefan Koch (ed.), Free/Open Source Software Development, Idea Group, Inc.

Aoyama M, 2002, "Metrics and Analysis of Software Architecture Evolution with Discontinuity", Proc. 5th Intl. Workshop on Principles of Software Evolution, IWPSE 2002, Orlando, FL: 103 – 107

Box G.E.P et al, Statistics for Experimenters, An Introduction to Design, Data Analysis and Model Building, Wiley, 1978

Brooks F, 1995, *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Ed., Addison-Wesley.

Capiluppi A., 2003, "Models for the Evolution of OS Projects", Proc. ICSM, Amsterdam, 22 – 26 Sept. 2003, pp. 65 - 74.

Capiluppi A., Morisio M. & Ramil J.F., 2004a, "Structural Evolution of An Open Source System: A Case Study", Proceedings of the 12th International Workshop on program Comprehension(IWPC), June 24-26, 2004, Bari, Italy, pp. 172 - 182 .

Capiluppi A., Morisio M. & Ramil J.F., 2004b, "The evolution of source folder structure in actively evolved open source systems", Proceedings of the 10th International Symposium on Software Metrics, Sept. 11-17, Chicago, Illinois, pp. 2 - 13

Capiluppi A., Faria A.E. & Ramil J.F., 2005, "Exploring the Relationship between Cumulative Change and Complexity in an Open Source Systems", 9th European Conference on Software Maintenance and Reengineering (CSMR), Manchester, UK, March 21-23, 2005

Dalle J.M., David P.A. 2004. *SimCode: Agent-based Simulation Modelling of Open-Source Software Development*, available online at http://opensource.mit.edu/papers/dalledavid2.pdf <as of Feb. 2005>

FEAST, 2005, *Feedback, Evolution And Software Technology*, Dept. of Computing, Imperial College, http://www.doc.ic.ac.uk/~mml/feast/ <as of Oct. 2005>

Godfrey M and Tu Q, 2001, "Growth, Evolution and Structural Change in Open Source Software", Proc. of the 4th Intl. Workshop on the Principles of Software Evolution, Sept. 10-11, 2001, Vienna, Austria.

Fischer M, Pinzger M and Gall H 2003, "Populating a Release History Database from Version Control and Bug Tracking Systems", Proc. ICSM 2003, 22-26 Sept, Amsterdam,: 23 – 32

Herraiz I, 2005, "The Evolution of Large Open Source Software", research seminar, Computing Dept., The Open University, Milton Keynes, U.K., 28 July 2005.

Iwasaki, Y. and Simon, H. A., 1986, Causality in Device Behaviour, *Artificial Intelligence* **26**: 3-32.

Lehman M.M., 1974, "Programs, Cities, Students, Limits to Growth?", Inaugural Lecture, in *Imperial College of Science and Technology Inaugural Lecture Series*, vol. 9, pp. 211 – 229.

Lehman M. M. and Belady L 1985; *Program Evolution – Processes of Software Change*, Acad.

Press, London, 1985. Available from links at: http://w3.umh.ac.be/evol/publications.html <as of October 2005>

Lehman M.M., 2000a, "Approach to a Theory of Software Process and Software Evolution", FEAST 2000 Workshop, Imp. Col., 10- 12 Jul. 2000, also as Res. Rep. 2000/2, Dept. of Comp., Imp. Col., Feb. 2000 http://www.doc.ic.ac.uk/~mml/feast2/papers/pdf/645.pdf <as of October 2005>

Lehman M.M., 2000b, "Rules and Tools for Software Evolution Planning and Management", FEAST 2000 Workshop., Imp. Col., 10-12 Jul. 2000, also as Tech. Report, 2000/14 Nov. 2000, Imp. Col., Dept. of Comp, http://www.doc.ic.ac.uk/~mml/feast2/papers/pdf/611_2.pdf <as of October 2005>

Lehman MM and Stenning V., 1996, "FEAST/1 Proposal Part 2, Case for Support", EPSRC proposal, Computing Dept. Imperial College London, 1996, http://www.doc.ic.ac.uk/~mml/feast2/case96-2.html <as of Oct 2005>

Lehman MM, Kahen G and Ramil JF, 2002, Behavioural Modelling of Long lived Evolution Processes– Some Issues and an Example, *J. of Software Maintenance and Evolution* **14**: 335 – 351

Madhavji N, Lehman M, Perry D, Ramil JF, (eds), 2006, *Software Evolution and Feedback*, Wiley, 2006.

Madey GR, Freeh VW, Tynan RO, 2002, "Agent-Based Modeling of Open Source using Swarm", Proc. of Americas Conference on Information Systems (AMCIS 2002), Dallas, Texas, August.

McCabe TJ, A complexity measure, 1976, IEEE Transactions on Software Engineering, SE-2 (1976) pp. 308-320.

McCabe TJ and Butler CW, 1989, Design Complexity Measurement and Testing, *Communications of the ACM* **32**(12): 1415 – 1425

Mens T Ramil JF, and Godfrey M, 2004, Analyzing the Evolution of Large-scale Software: Guest Editorial. *Journal on Software Maintenance and Evolution* **16**(6): 363-365.

Mockus A., Fielding RT, and Herbsleb J, 2002, Two Case Studies of Open Source Software Development: Aache and Mozilla, *ACM Trans. Software Engineering and Methodology* **11**(3): 309-346.

NetLogo, 2005, http://ccl.northwestern.edu/netlogo/ < as of Oct 2005>

Rajlich VT and Bennett KH, 2000, A Staged Model for the Software Life Cycle, *IEEE Computer* **33**(7): 66 – 71.

Ramil J.F. & Smith N., 2002, Qualitative Simulation of Models of Software Evolution, *Journal of Software Process: Improvement and Practice*, **7**: 95 – 112.

Raymond, E. S., 2001 *The Cathedral and the Bazaar* O'Reilly Media Inc.

RELEASE, 2005, REsearch Links to Explore and Advance Software Evolution, http://labmol.di.fc.ul.pt/projects/release/ <as of Oct. 2005>

Robles G., Merelo J.J., Gonzalez-Barahona J.M. 2005 "Self-Organized Development in Libre Software: a Model based on the Stigmergy Concept", ProSim 2005, St Louis, Missouri,May 21-23, 2005.

Rocha, 2003 "Complex Systems Modeling", Indiana University and Los Alamos National Laboratory, http://informatics.indiana.edu/rocha/complex/csm.html <as of Oct 2005>

Scacchi W, 2006, Article on the evolution of open source software to appear as a book chapter in Madhavji *et al.* (2006).

Smith N., Capiluppi A., Ramil J.F., 2005, A Study of Open Source Software Evolution Data using Qualitative Simulation, *Software Process Improvement and Practice* **10**: 287-300.

Sommerville I, 2001, *Software Engineering*, 6th Ed., Addison-Wesley, Wokingham, UK.

Turski W.M., 1996, A Reference Model for the Smooth Growth of Software Systems, *IEEE Trans. Softw. Eng.*, **22**(8): 599 – 600

Turski W.M., 2002, The Reference Model for Smooth Growth of Software Systems Revisited, *IEEE Trans. Softw. Eng*., **28**(8): 814 – 815.