Nathan Oostendorp
April 27, 2009

An Exploration of Open Source Software Contribution Networks Using the SimForge Agent-Based Model

Abstract

While a great deal of research has explored the micro-level incentives for developers for contribution to Open Source Software, the field of innovation research lacks a generalized macro-model for how Open Source developers interact within a software ecosystem. In this paper, I propose a model consisting of Developers as self-interested agents who find and improve software based on discrete preferences for software represented as bundles of features. This "SimForge" model can create networks of developer and software relationships which are modeled after real developer/software networks found on the site SourceForge.net. While this model is exploratory, it shows a potential for a deeper understanding and modeling of Open Source Developer behaviors.

Introduction

In the last two decades the Open Source Software movement has produced a large number of robust software solutions thorough an open, collaborative method of decentralized expertise and effort. In forums such as the Open Innovation Workshop [1] interest has been building in the organizations, both formal and informal, that form around open source software projects. In this paper, I illustrate an Agent Based Model called "SimForge" which uses some basic principles and a self-interested set of agents to "grow" an Open Source ecosystem based on Developers desire to use their computer for specific tasks, and the software they can create and improve collaboratively.

I will start with a very brief survey of previous work looking at the formation of Open Source supporting social systems, and describe some of the assumptions I have made in creation of the model. I will then illustrate the model in detail. I will also outline some of the results I observed in some experimental sweeps of the model's parameter space. Finally, I will make some comparisons with output from this model and real data from the website SourceForge.net, and look at how this model could be improved, and where it is structurally incompatible with real data.

SimForge is an exploratory Agent-Based model, and so it is unlikely to withstand strenuous validation tests or have strong predictive value. It is intended to be more of a rigorous thought experiment, looking at how a very simplified world of self-interested software developers could create a library of open software collaboratively. I hope that this may inform development of future models, and may aid in making sense of the formation of the Open Source Movement in general.

Prior Work

The concepts behind SimForge draw from a good deal of research and theory about Open Source software which has been developed over the last decade. In one of the earliest, Lakhani and Wolf [2] administered a survey of several hundred developers working on a variety of software projects on SourceForge.net. They found that many of the developers

were acting largely on intrinsic motivators, and that the description of these motivations varied substantially from person to person. Their work substantially clarified thinking around the open source movement, helping to discern "Why do people build software for free?" Since often when the developers felt that they had an enjoyable creative outlet, they were involved in something tha created both value for themselves intrinsically as well as value for users of the software. In SimForge, we control this overall amount of "motivation" in the system through a single static parameter, a global "desire to contribute".

In order to understand the interactions of self-interested agents and Open Source, we look specifically at economic models where software is a resource in a repeatable game, in this case between firms. Bessen [3] describes an environment where firms either create software, and then seek to either develop the software collaboratively with other firms, or extract rent by licensing the software as a monopolist. Bessen's model includes two major design components which I have incorporated into SimForge, notably the expression of software as bundles of discrete features, and the concept of independent agents who can rationally choose to collaborate and create software in a shared space. This model also includes reference to "complexity overhead" costs while developing software -- the property of software that as it becomes more complicated and complex, it requires more effort to create additional value using that technology.

In terms of objective, SimForge most closely resembles the work of Madey et al [4] [5] [6]. Their work uses an agent based model in which developers "grow" software contribution networks which resemble real SourceForge.net data. In the model, developers can start new projects, join other projects, and quit projects and are attracted to projects based on the quantity of developers already working on it. SimForge can be considered an extension of this model, but is distinct in that rather than the developers forming groups based on the attractiveness of the other developers, the software contains specific features that developers have intrinsic preferences for. These features and their value against developer preferences dictate attachment logic, rather than the social system surrounding the software being an attractive force.

Simplifications in SimForge

In order to make a easy to understand model, we have reduced the number of potential features into six major parameters. In order to do this, it is necessary to "boil down" a good deal of the complexity known to exist in the real world of Open Source Software. Here are a brief summary of a few of the points I decided to leave out of the model, but which could be integrated in future research.

Developers level of expertise and effort is uniform - While realistically open source developer commitments vary widely from dabblers to full time programmers on Open Source projects, the SimForge developers are controlled entirely by random chance based on a single uniform parameter. This creates a binomial distribution of participation centered around the desire to contribute parameter.

Developers act on primitive incentive structures - in SimForge, we expect a developers work on a software project to maximize their own value on software, and so traditional extrinsic motivators such as reputation building, employment, or contracting are not in play.

Software's value is entirely represented in its features - the value of software is discerned specifically based on the features it implements and the amount of effort exerted in development. The software does not exhibit "network effects" or any other externalities of use.

Developers show no social preferences for working with each other - the Developers choose to work on Software, not with other users

All Software is completely democratic in development - Any Developer can improve any feature in a Software project, and software organizations show now difficulty growing to large numbers.

Tool chains, Software Libraries, and Programming Environments do not exist - Software in SimForge is developed completely independently of any other software, so there are no interdependence between Software projects in the system.

Description of the Model

In the model, Developers act as self interested agents who have an array of potential tasks that they wish to accomplish with their computer. These are called "Features" and can be construed both as realistic software features, or as any bundle of functionality which would be required to accomplish an arbitrary task. The developer's relative value for each feature is used as a probability distribution from which they choose a single feature in each round, so if she has a value for feature 1 at 7 and feature 2 at 1, feature 1 is 7x more likely to be a desired activity in any given step. If no software implements that feature, they may choose to create it. If software exists for that feature, a developer may decide whether or not to contribute. If they decide to contribute, then they will extend the software in a way which maximizes the value for them -- for example, a user with a high value for Web Browsing and a low value for Email might choose to add web browsing to an existing email client if they were using email that turn. The decision logic for developers is outlined in Figure 1.

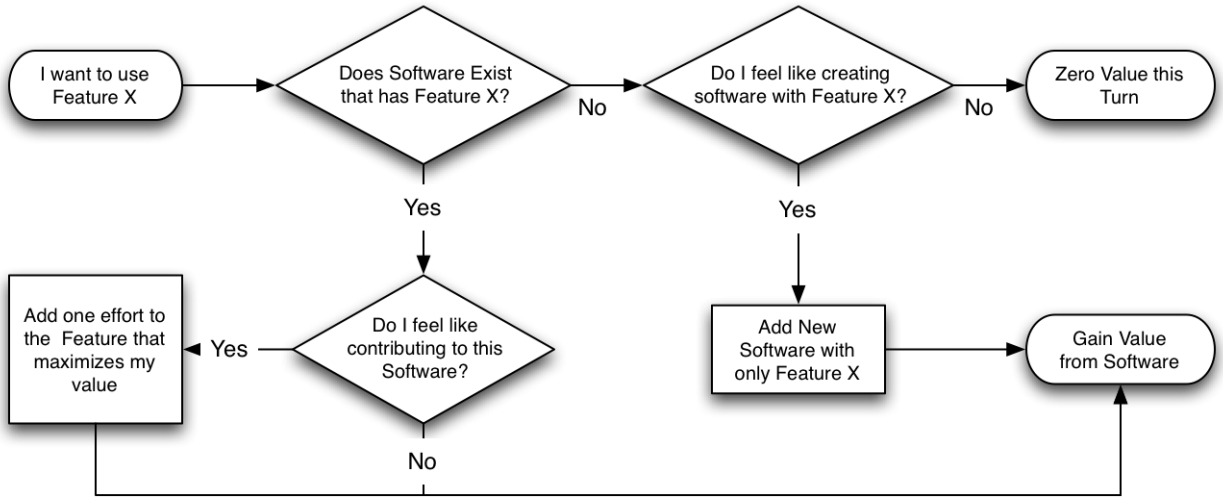

Figure 1: Logic of Developer Agent

The Software which the developers build is represented as a bundle of features with discrete values, which the developers can improve and extend. Each turn, if a developer chooses to extend a piece of software, they may either improve a feature already in the software or add a new feature to the software. The amount of effort developers put into software is uniform, so any developer effort adds linearly to that feature's value in the software. However, I have added two dynamics which bound the *depth* as well as the *breadth* of software development, and this keeps a piece of software from becoming so valuable on a single attribute that developers always select it, and likewise prevent a single piece of

software from implementing all features.

In order to curtail the amount of effort dedicated into a single feature, we use a **sub-linear value function** for features so that the worth extracted from that feature equal to $LOG_2(1 + \text{preference strength} * \text{effort})$. This means that early development in the feature creates the most value, while subsequent development increases value further, but at a declining rate. Similarly, in order to curtail the number of features software can contain, a **Complexity Penalty** parameter was added. This reduces the effect of effort on software by a multiplying factor of $1 / (\text{Number of Features}^{\text{Complexity}})$. Since this is adjustable, a Complexity value of 0.0 means no penalty at all, while at 1.0 the second feature doubles the amount of required effort and the third triples it. At a Complexity of 2.0, a second feature makes the software 4x as difficult to develop, and a third feature 9x harder, a fourth 16x, etc.

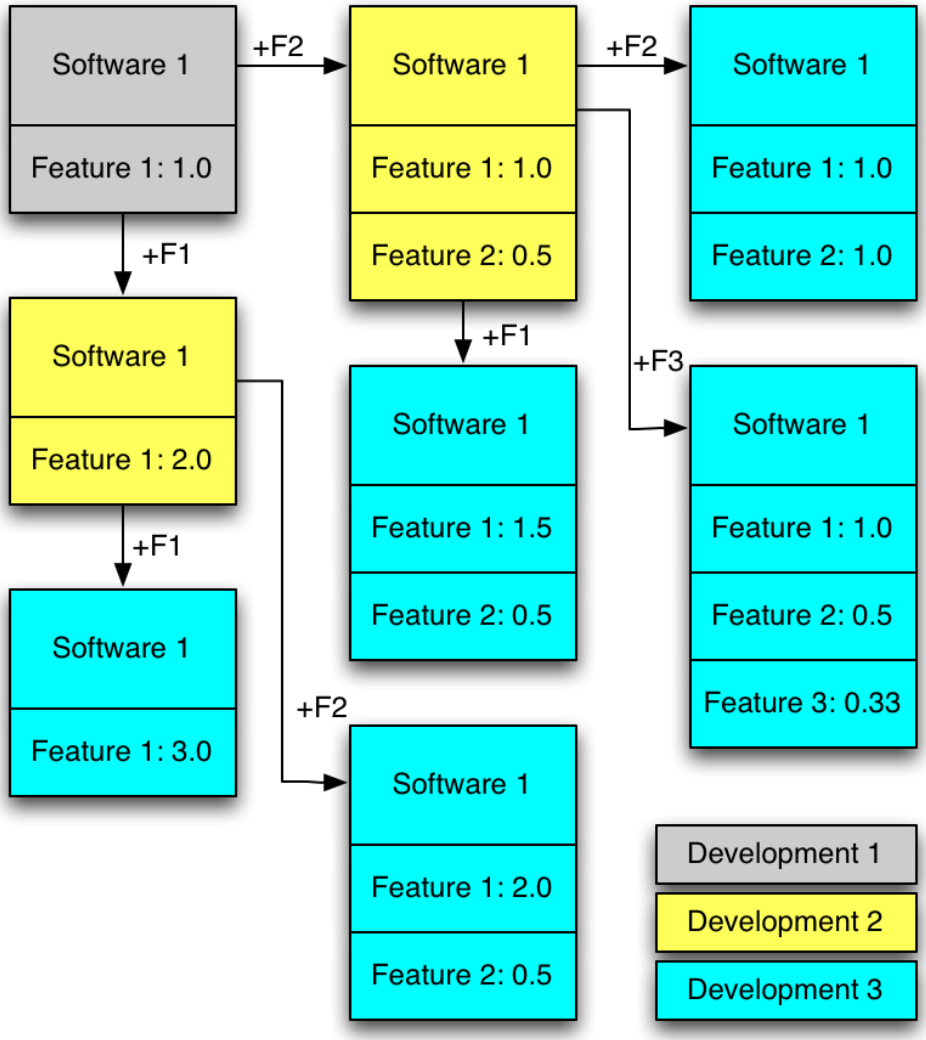

Figure 2: Three Rounds of Software Development At Complexity Penalty = 1.0

Figure 2 shows an exhaustive example of a piece of software being developed over 3 rounds in an environment where Complexity is set to 1.0. In round one, it has a single feature with 1.0 effort allocated to it. In round two, a developer could choose either to improve Feature 1 to 2.0, or to add a new feature. With this new feature, complexity-related costs are incurred, so it can only be improved to 0.5. By round 3, we have a number of potential

outcomes. In the broadest of these, with three features, the complexity costs have now become 3x so Feature 3 could only be developed up to 0.33. Alternatively, in the deepest outcome, Feature 1 remains in focus and gets 3.0 effort. The other outcomes lie somewhere in the space between these extremes.

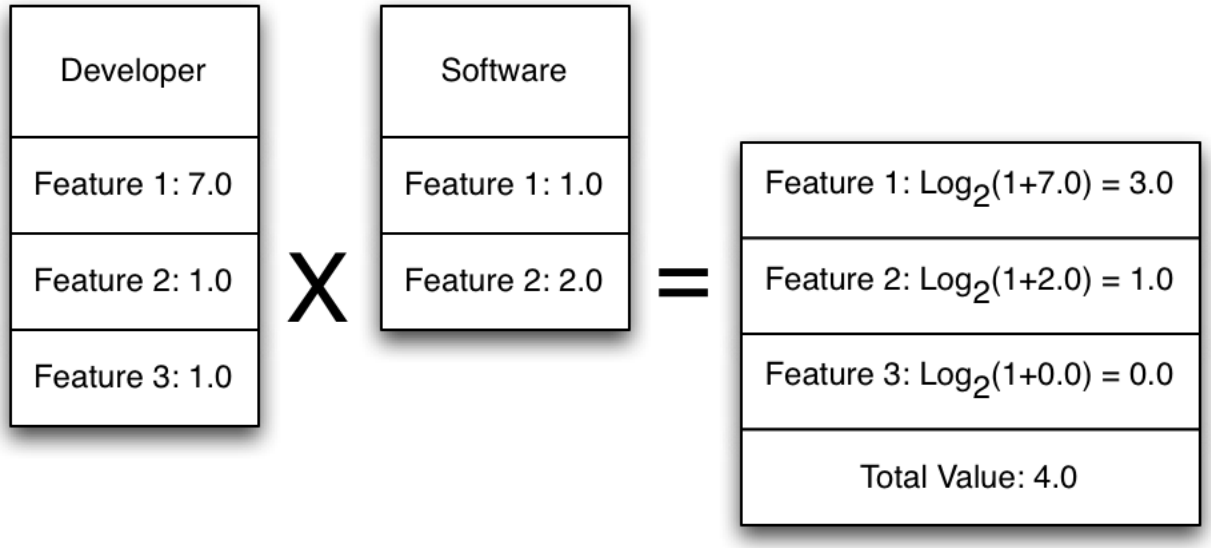

Figure 3: Calculation of Developer Value for Software

Figure 3 illustrates how a developer's preference for different features and the amount of effort which has been input into the software's features are used to calculate the value of the software to the developer. In this case the developer has a high preference weight on Feature 1, and low weights on features 2 and 3. The software he is using has a value of 1.0 on Feature 1, and a value of 2.0 on Feature 2. When we calculate the value, we take the product of the preference weightings and the feature values in the software. Each of these are scaled by Log_2 and the sum of the feature products is the total value, which in this case is 4.0.

While these depth and breadth-bounding features add complications to the model, they are meant to represent constraining forces which are very real in the development of software. Bessen [3] in particular discusses how the complexity penalty may be closer to exponential, being $2^{(\text{Number of Features})}$ and refers to the widely known 80/20 rule of software, that the final 20% of "edge case" functionality takes up 80% of the time.

Feature Preference Assignment to Developers

In SimForge, each developer has some positive value for every feature in the system. Two parameters in the model control how developer preferences are assigned to each developer. One is a **pareto factor** which controls the amount of "focus" developers have on different features in the system, by controlling the "long tail" distribution which developer preference values are drawn randomly from. When the pareto factor is high, the developer shows a very balanced appetite for all potential features, but as the pareto factor falls the preference weights become more focused on individual features. Figure 4 shows two developer preferences at different Pareto factors, and on the lower value the disparity between preferences is apparent.

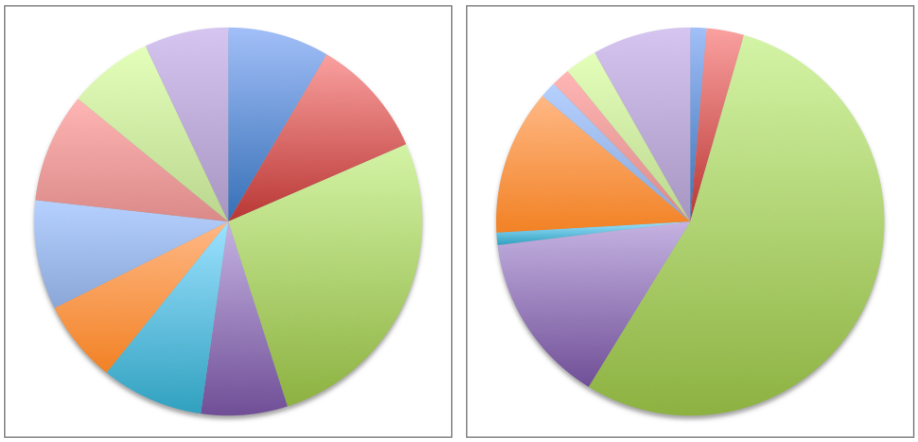

Figure 4: Developer Feature Preference Weights for 10 Features, (a) Pareto = 1.8 and (b) Pareto = 0.6

The other parameter which controls developer preferences is a **Homogeneity Factor**, which controls how similar the preferences are between developers. The factor itself is a value between 0 and 1 which represents a uniform distribution threshold, and each feature weight for each developer has that much chance of receiving a preference from a "global preference profile". At 0, all developers select completely randomly from the specified Pareto distribution, and at 1, all developers are the same, but their relative preference weights still follow the Pareto factor's curve. Figure 5 shows a population of 4 developers with a high homogeneity factor. In this population you can see how the largest weighted factor is often the light purple wedge on the right, but preference weights are similar but vary among the developers.

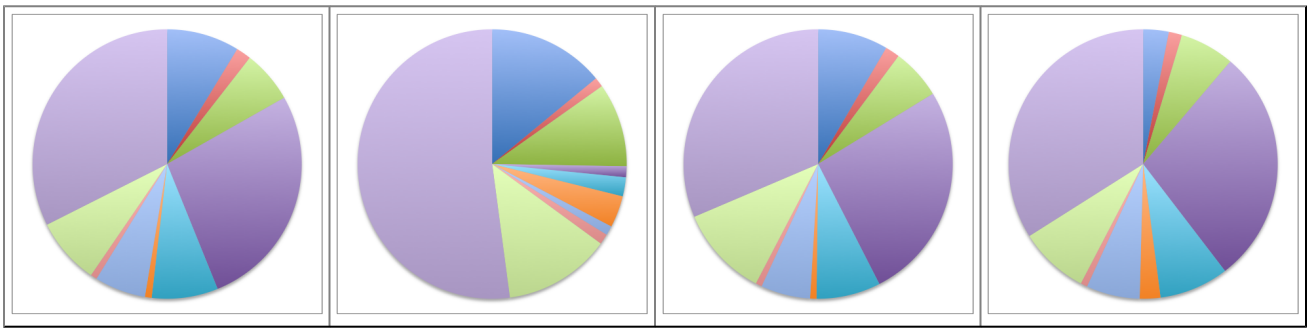

Figure 5: Four Developer Feature Preferences Profiles for 10 Features at Pareto = 1.3, Homogeneity = 0.75

Model Parameter Space

The basic parameters controlling SimForge are outlined in Table 1, and represent the dimensions in the space manipulated in direct experimentation. I divide these parameters between **Quantitiative Network Controls**, on top, which control how many nodes of each type and how many edges exist in the system, and **Preferential Attachment Controls** which define how Developer nodes form edges within the network. In the experiments, I focus largely on the interaction and effect of the Preferential Attachment group, while fixing

the amount of "stuff" in the system.

Parameter Description	Representation for Model	Observed Network Effect
Quantitative Network Controls		
The *Number of Developers* active at turn 1 of SimForge and the *Number of Developers Added* each turn to SimForge	The population and growth of contributors to the open source community	Controls the number of developer nodes and rate of new developers in the system
Probability Developer will *Desire to Contribute* to Software	Factors influencing decisions to engage in open source contribution	Controls edge weight and density
Number of Features for creation of Software	Represents how many different properties of software affect developer preferences for use	Controls number of software nodes in the system, likelihood of collaboration
Preferential Attachment Controls		
Pareto Factor (developer focus) for Developer Preferences	This random draw from a pareto distribution represents the "long tail" popularity of software	At high values, approaches random attachment, at low levels developers are focused on one or two features.
Homogeneity Factor (developer similarity) for Developer Preferences	This expresses how people's software preferences are often "alike", so "web browsers" and "word processors" are more frequently used than "network analysis tools"	Correlation of attachment for developer nodes
Complexity Penalty for Software Features	This represents how it gets harder to develop complicated software packages, expressed exponentially	Weak effect on number of Software nodes, effect on average features per Software

Table 1: Description of Parameters

Model Implementation Details

My implementation of SimForge has been published on SourceForge.net [7], and an interactive version of the model is publicly available at http://simforge.oostendorp.net . The system was implemented in Python using a number of external libraries, and the Django web framework for the web interface, which is shown in Figure 6. The windows shown, clockwise from top left, show a graph representation of the network, with developers in red, and software in blue. A form for changing parameters in the model, graphs of time-series statistics, and logfile output from the models execution. This system allows for very cursory exploration of the model parameter space, although unlike typical Swarm/Repast models it does not include a "step"-wise function.

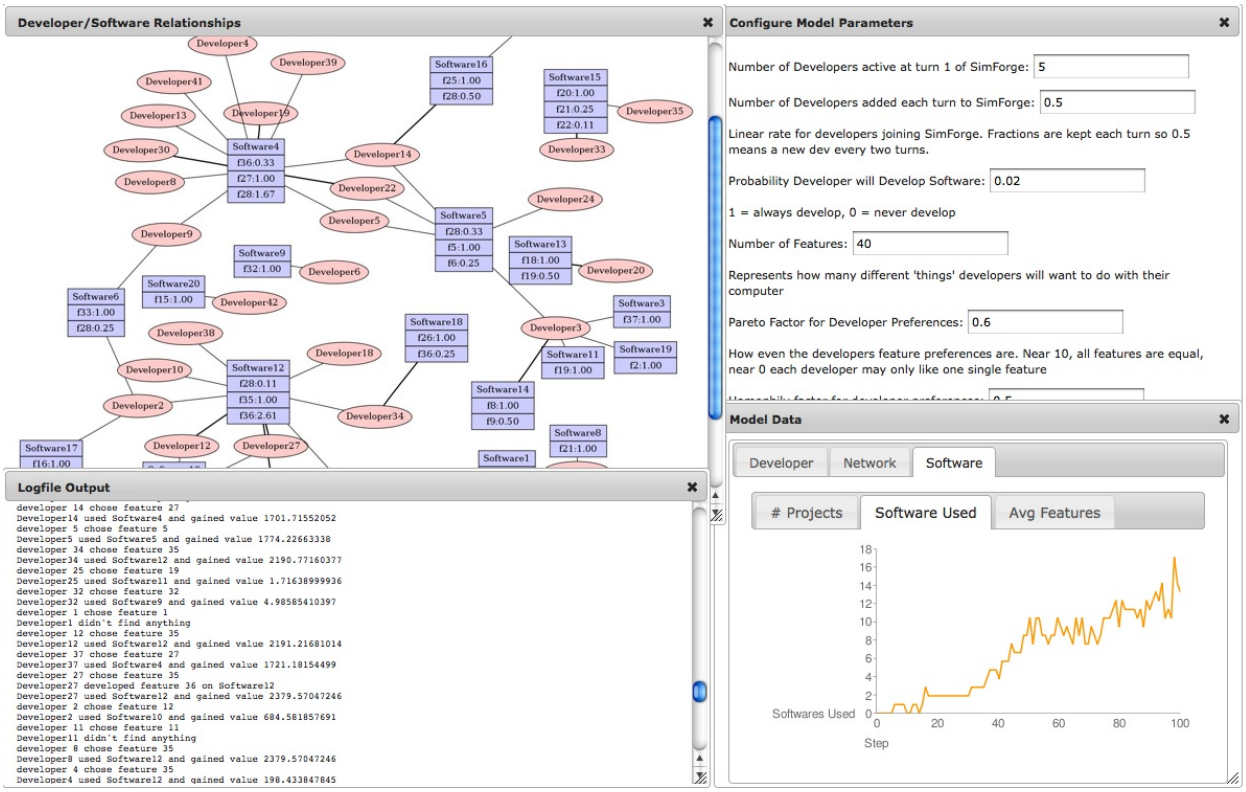

Figure 6: Web Interface for SimForge

For repeated experimental runs the code is accessable via command line with configurable parameters. My experiments ran on a 5-instance Eucalyptus cluster [8] with 10 cores running simultaneously. Each of my experiments I fixed the number of Developer and Features, and ran 10-step ranges for free parameters, with 10 repetitions of each specific parameter set. The specific parameter ranges for each experiment are shown in Table 2 -- Experiments 1 and 2 both consisted of a total of 100,000 iterations and Experiment 3 added a rate of growth to the developer population, totaling 1,000,000 iterations. This final experiment took roughly 16 hours of time on the cluster.

Parameter	**Experiment 1**	**Experiment 2**	**Experiment 3**
Number of Developers	50	50	5
New Devs Per Turn	0	0	0.01 - 0.2
Number of Features	50	50	50
Desire to Contribute	0.01 - 0.1	0.01 - 0.1	0.01 - 0.1
Number of Turns	200	200	200
Complexity Penalty	0.2 - 2.0	0.2 - 2.0	0.2 - 2.0
Pareto Factor	0.1 - 10.0	0.1 - 2.6	0.1 - 2.6
Homogeneity Factor	0.0 - 1.0	0.0 - 1.0	0.0 - 1.0

Table 2: Ranges Tested in Experiments

One of the shortcomings of my implementation is the amount of computation required per turn is quadratic with the number of developers * the number of features. Therefore it was

impractical to me to attempt to simulate an environment as large as SourceForge.net, which would have hundreds of thousands of both features and developers. Future work could look at refining performance or implementation in a non-interpreted language.

Results of Parameter Range Experiments

This initial exploration revealed the sensitivity of several statistics to parameters using a small developer population (less than 50). Since the "Desire to Contribute" parameter is a static probability each turn, it can be used as an instrument to look at different places in time, since at large numbers of steps even minuscule probabilities would be represented. For most of the results, Desire to Contribute was fixed at its median range value to roughly fix the number of edges present in the system for comparison.

Of the experiments listed in Table 2, Experiment 1 was purely exploratory, and I discovered that the initial range for the Pareto Factor for preferences was far too wide. In fact there was very little variability in most statistics for otherwise identical runs above a Pareto Factor of 2.5. Experiment 2 was executed to give a more detailed picture of the most interesting portion of this parameter. Experiment 3 was run to look at the overall effect of different rates of growth in the system. Below are some of the results I found most interesting about the space, particularly looking at cross sections where I expected strong interaction.

Result 1: Complexity Penalties have a Significant, but Weak Impact on Number of Software Projects and Average Number of Features per Software Project

When designing this model, I had some concerns about the Complexity Penalty being far too strong because of its exponential nature. I expected that low complexity-penalty environments would have fewer Software projects, and that these projects would have more features. While this turned out to be the case, Figure 7 shows that this interaction is quite weak, accounting for small fraction of overall variation. These measurements were generated from experiment two with the Pareto and Homogeneity factors as free parameters, but with the number of nodes and number of software contributions roughly fixed.

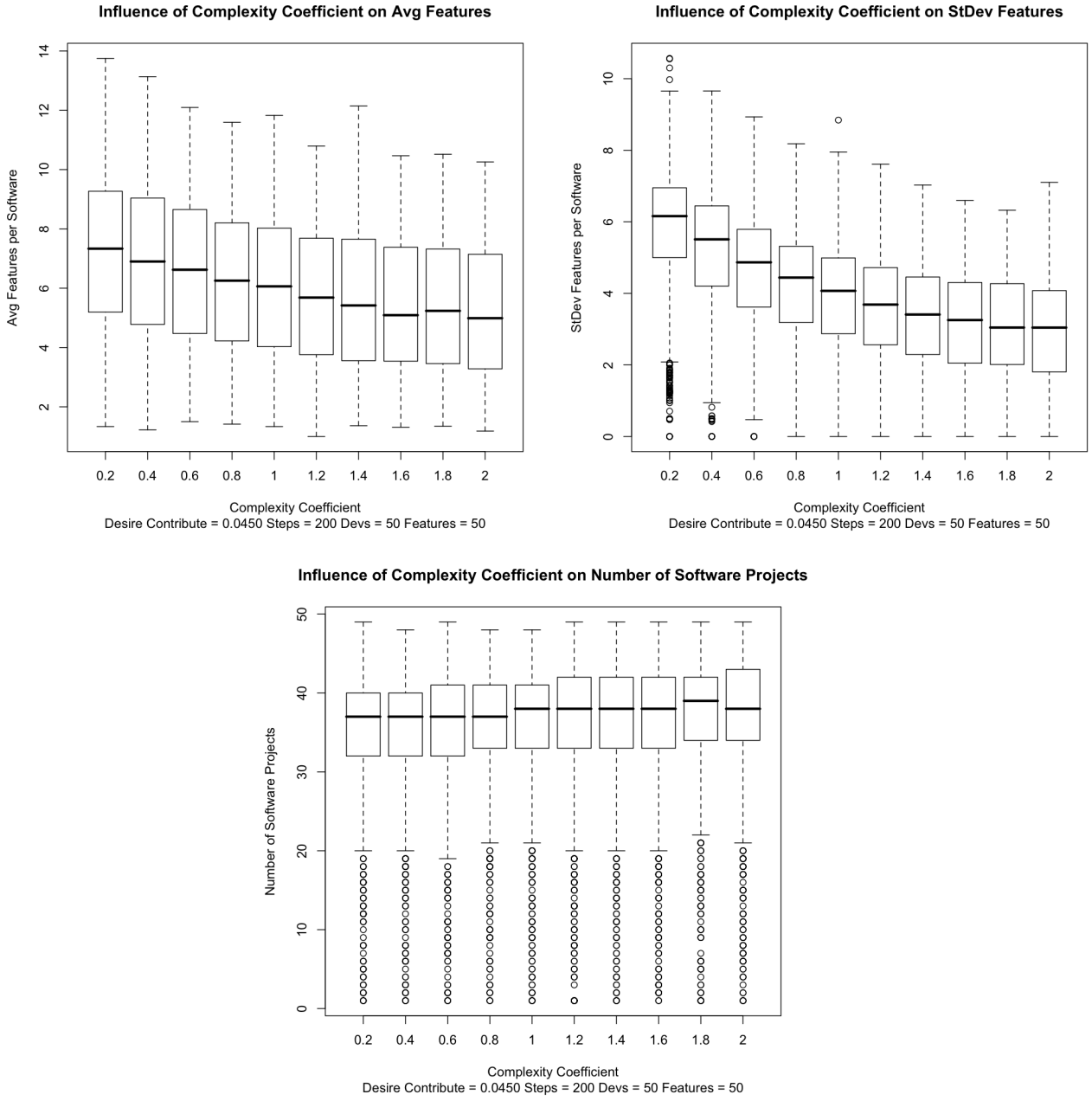

Figure 7: Influence of Complexity Penalty on (a)Mean, (b)Std Dev of Features per Software and (c)Number of Software Projects (Experiment 2)

While there is a statistically significant impact, from my lowest measured value of the parameter, 0.2 (nearly linear effort to value penalty) to the highest, 2.0 (quadratic effort to value penalty) on average only 1.7 fewer features were implemented per software project, the mean standard deviation of features narrowed by 2.6 features, and on average only 0.97 more software projects. This may be the result of not exploring extremely high values in the parameter space, but it also may be that the scale-free nature of preferences combined with the sub-linear value function on feature effort force software projects to diversify, even under challenging conditions.

Result 2: The Homogeneity Factor has a Surprising Distribution of Inequality

One of the statistics that I tracked as part of these experiments was a Gini coefficient [9] which is used to measure inequality among agents. In this case, I wanted to see what the overall value to for each agent at the end of the iteration, and compare it to the other developers. I expected that with a low Pareto value, inequality would be high, since some agents who happened to share a strong interest with another person, or created their software early in the simulation would reap a windfall, while others may find software which implemented the feature as an afterthought. My other hypothesis was that heterogeneous populations would have higher inequality, and highly homogeneous populations would show very low inequality. Figure 8 shows how that is largely correct, except that at low homogeniety there is an increase in inequality over the minimum value.

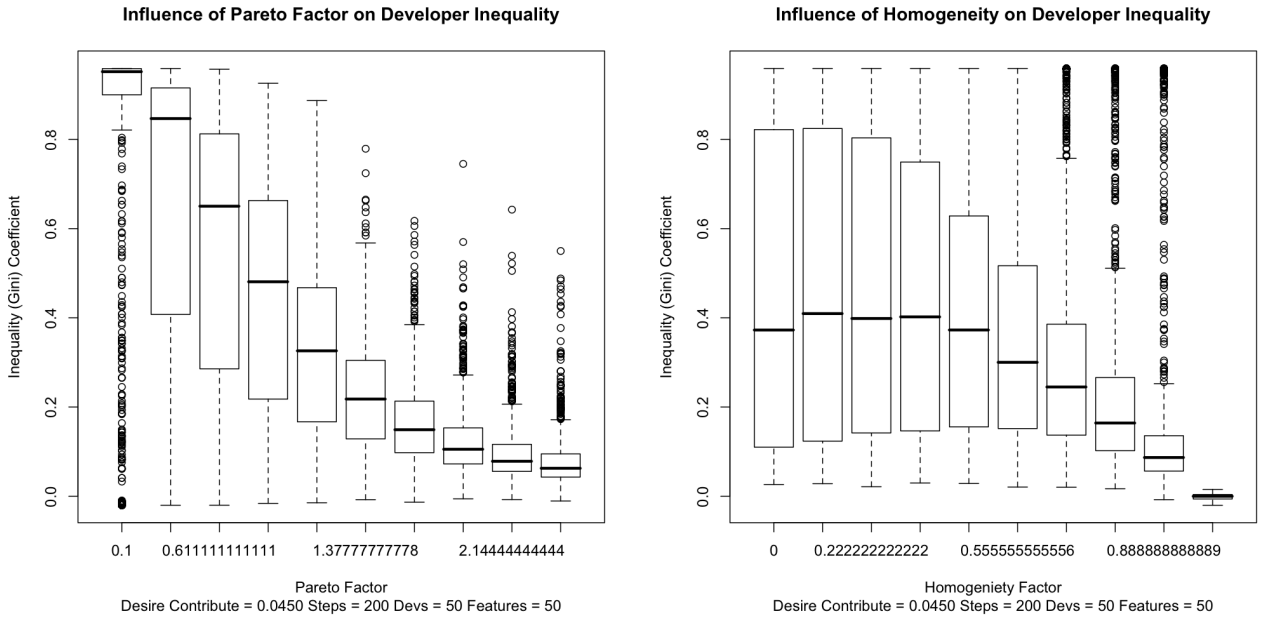

Figure 8: (a) Pareto Factor and (b) Homogeneity Factor impact on Developer Inequality (Gini Coefficient)

While this was not a result that I expected, I would possibly explain it by my particular implementation of homogeneity, which was using a global preference profile. In the case where each preference value has a low probability of being taken from a global profile, agents who are "lucky" enough to get a very high preference value will be more likely to collaborate than other agents. When this probability gets higher, this still happens but more "lucky" agents spreads this phenomenon out more, so that there is less inequality as measured by Gini. While this result is not tremendously strong (at 95% confidence an increase of Gini of 0.07 from homogeneity = 0), it is statistically significant, and an interesting outcome of the model.

Result 3: Graph Connectedness is Largely Controlled by the Pareto Factor (preference focus) Parameter for Developers

Another statistic which I kept track of was the amount of the network which was fully connected, that is, the relative size of the Giant Component (largest connected portion) vs the entire system. This is of interest to me, since SourceForge.net's membership network has a giant component which makes up 8.3% of the total network [10]. If this model was to be elaborated, it would be useful to incorporate that parameter into real data. What I

found was that the Pareto Factor controlled almost all of the variability for that statistic, with some slight impact from the Homogeneity Factor. Figure 9 illustrates their relative impact.

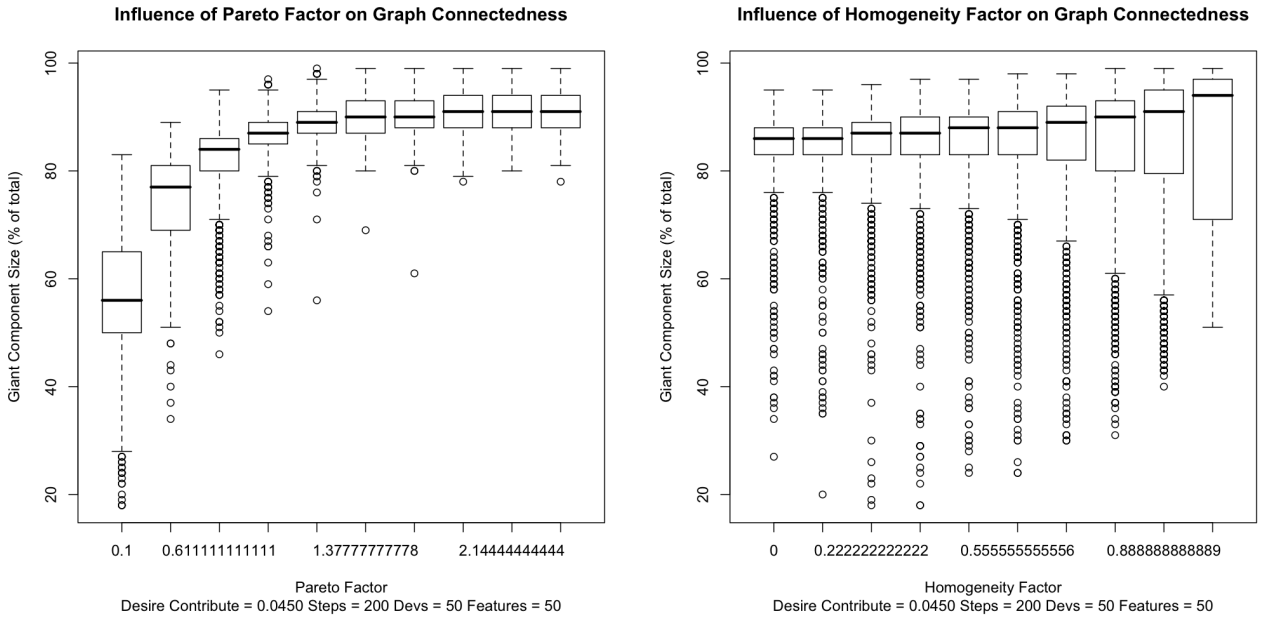

Figure 9: The Impact of (a)Pareto Factor and (b) Homogeneity Factor on Relative Size of Giant Component

While I had expected the combination of high Pareto Factor (random attachment) and high Homogeneity Factor to produce a well-connected network, it was interesting to me that the Pareto Factor seemed to control most of this dynamic. This would indicate within the context of this model that SourceForge developers would have a very low Pareto Factor and therefore a very high focus on single features, and that a Pareto curve for preferences could be an inappropriate distribution to fit this. However, within the model for a realistic fit of data more exploration should be explored in a very low Pareto range.

Result 4: Network Growth by Developer Entrance Has a Minimal Impact on Final System State

In Experiment 3, I attempted to see what different rates of growth had on the system as a whole. However from my evaluation, the entrance of new developers did not have a strong impact on the dynamics of edge formation in the system. All of the relationships between the Preferential Attachment Parameters and their impacted statistics seemed to be preserved, albeit with some scaling for the volume of nodes in the system and otherwise no noticeable change. It therefore seems somewhat irrelevant whether the nodes in the system are fixed or enter gradually in terms of final outcome.

While I found this result somewhat surprising, I can justify it somewhat by thinking about agent creation at a linear rate as equivalent to a uniform distribution of the desire to contribute parameter over a specific range. Under a system where contribution was set at a fixed 2% probability, over 200 steps a developer entering at round 1 would have an average of 4 contributions, whereas an entrant at round 100 would have only 2. Since in this case I am only looking at the end state of the system, the latter entrant could potentially have started at round one, but have had a 1% probability of contribution each turn. While this might have some impact on connectedness in high Pareto Factor (random attachment)

circumstances, under a strong preference focus (low Pareto) it would be unlikely to change the edges created by that developer.

Discrepancies between SimForge and SourceForge.net Data and Future Work

Initially I had designed this model with the intention that it could be parametrized using real data from SourceForge.net, however there are a number of issues which need to be addressed in the model before it could be robust enough to fit real data. The most obvious to me is that a static uniform probability for contribution is too primitive to reflect the network structure of SourceForge, since in the actual network the edge distribution of project contributors is close to scale-free [10]. In addition, to make this network look realistic in through time as well as at the final configuration, a fairly complex system for determining developer entry and exit would have to be devised. Many projects see "bursts" of development which are not present in this model since software has a constant rate of activity.

One tactic which I briefly explored was decomposing the network into size 2 m-slices by developer (clusters of software which shared 2 or more developers) and seeing if the model could produce reasonable networks which resembled these clusters. I began work with 6 clusters which I had used for a visualization project, and from which I had already had several statistical metrics computed [10]. While it was likely possible to control outcome of those same statistics in my model to match these clusters it seemed it would be an obvious overfit to make any claims about this fairly simple model on a few specific instances. I do think for future work, a more macro-level approach using multiple clustering methods could be employed, so that different "niches" of development could potentially be identified as having distinct attributes according to model-fit parameters.

I also believe that the representation of features as a discrete array is somewhat limiting and artificial. At one point, I had conceived that features would exist in a "tree"-like structure, with rudimentary features (analogous to File and Network IO, or Windowing API calls) having to be developed before others could build on top of them. I think this might produce other interesting network properties and has certainly had an effect in the real Open Source community as development has grown from production of mostly system tools and network services, to building desktop applications, programming languages, and advanced networked software systems. This might be able to control some of the elements of the system which I was attempting to achieve by growing the developer population at a linear rate.

Conclusions

Overall, I believe SimForge has value as an exploratory model of an Open Source ecosystem, as it connects ideas from a variety of sources into an incentive-compatable system. While it is not sophisticated enough to mimic real data, it does illustrate how this open source can "grow" out of a decentralized network of individual developers. It is also a useful framework for thinking about how developers decide how to contribute on a project, since different developers may have somewhat hetrogenous preferences, and that may dictate how they change open source code.

References

1. HBS MIT Open Innovation Workshop Website http://www.openinnovation.net/Conference/UOI2008/
2. Why hackers do what they do: Understanding motivation and effort in free/open source software K Lakhani, RG Wolf http://papers.ssrn.com/sol3/papers.cfm?abstractid=443040
3. Open source software: Free provision of complex public goods JE Bessen, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=278148
4. Agent-based Modeling and Simulation of Collaborative Social Networks, Greg Madey, Vincent Freeh, Renee Tynan, Yongqin Gao, Chris Hoffman *AMCIS 2003*, http://www.nd.edu/%7Eoss/Papers/AMCIS2003_proceedings.pdf
5. Modeling the F/OSS Community: A Quantitative Investigation, G. Madey, V. Freeh, and R. Tynan, *Free/Open Source Software Development* 2004 http://www.nd.edu/%7Eoss/Papers/BookChapter.pdf
6. Modeling and Simulation of the Open Source Software Community, Yongqin Gao, Greg Madey, Vince Freeh *Agent-Directed Simulation Conference* 2005 http://www.nd.edu/%7Eoss/Papers/ADS011_Gao.pdf
7. SourceForge.net: SimForge Project http://sourceforge.net/projects/simforge/
8. Eucalyptus Clustering Website http://eucalyptus.cs.ucsb.edu/
9. The Gini Coefficient on Wikipedia http://en.wikipedia.org/wiki/Gini_coefficient
10. Using Networks to Visualize and Understand Participation on SourceForge.net, Nathan Oostendorp, 2007. http://sfnetworkviz.sourceforge.net/