

AN APPROACH TO VERIFYING THE IMPLEMENTATION OF A
SIMULATION USING A DOCKING PROCESS

A Thesis

Submitted to the Arts and Letters / Science Honors Program
of the University of Notre Dame
In Partial Fulfillment of the Requirements
for the Honors Program

by

Jeffrey Goett

Dr. Greg Madey, Advisor

Department of Computer Science and Engineering

Notre Dame, Indiana

May 2004

AN APPROACH TO VERIFYING THE IMPLEMENTATION OF A SIMULATION USING A DOCKING PROCESS

Abstract

by

Jeffrey Goett

A docking process, a method of verifying a result by showing that it can be reproduced by a different method, gives scientists a process beyond typical verification techniques to ensure the clean migration of a simulation from one language to another. To investigate this project's topic, whether the virtual community hosted at the Sourceforge website obeys the Barabási-Albert model of social network formation, the group implemented a set of 4 simulations using the simulation package SWARM. With the emergence of RePast, a SWARM-like simulation package written entirely in JAVA, the group rewrote each simulation in RePast in order to take advantage of the benefits of a full JAVA implementation. By statistically comparing the output of multiple iterations from corresponding simulations, we found and corrected errors in the migration, ultimately increasing their confidence that the docked simulations had been reimplemented in RePast correctly. While not a perfect or complete example of docking, this paper shows the effectiveness of a docking process in verifying the reimplemention of a model.

1.0 INTRODUCTION

1.1 Docking as a Verification Technique

A docking process, an allusion to the “docking” of two dissimilar spacecrafts, is, most generally, a method for verifying the result of a calculation by showing that a different method can reproduce the same result.

Two different approaches to a physics problem illustrate how one can use the docking technique to verify the execution of a calculation and to validate the accuracy of the solution. A ball begins from rest to fall towards the earth from a height d above the surface. To determine the velocity of the ball at the surface of the earth, one can manipulate these mechanics equations to solve for final velocity in terms of d and the acceleration due to gravity, g .

$$d = \frac{1}{2}gt^2 \text{ \& } v_f = gt$$

yields

$$v_f = \sqrt{2gd}$$

To check this solution, the one can also solve the problem using energy equations.

$$PE_i = KE_f$$
$$mgd = \frac{1}{2}mv_f^2$$
$$v_f = \sqrt{2gd}$$

Since it is improbable that one made a different mistake using the energy equations but derived the same solution as the first method, one can have more confidence that this solution is correct.

This physics docking problem verifies that the equations were correctly “executed” and validates both the mechanics and energy models of the world.

Figure 1.1 illustrates this.

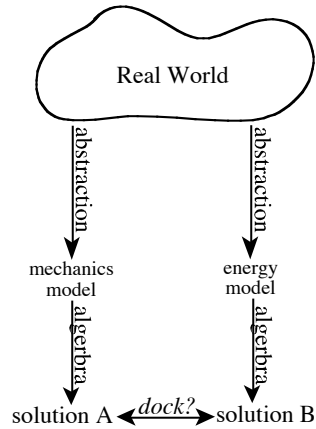


Figure 1-1:
Docking can both validate the corresponding models and verify that the algebra was done correctly.

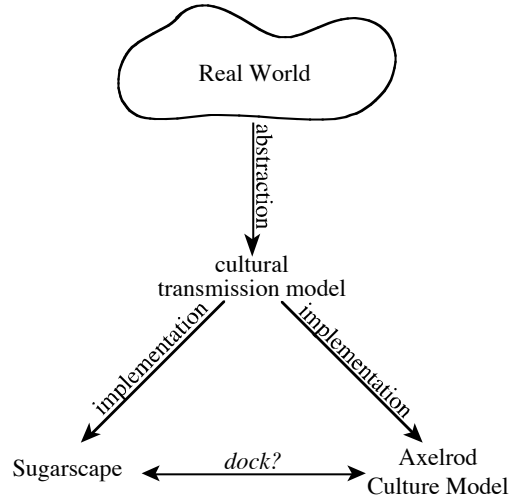


Figure 1-2:
Docking can verify the implementation of Sugarscape and the Axelrod Culture Model.

For scientists writing computer simulations, docking is another tool for verifying the implementation of their simulations.

Simulation verification determines whether a simulation correctly implements a given model. Tradition techniques for verification include checking whether output data plausibly corresponds to the expected output from the model. Tracking the value of simulation variables as the simulation runs can also reveal whether it executes as expected (Banks, 2001, 369-74).

Robert Axtell, et al. (Axtell, 1996) ran a docking experiment to verify the implementation of the cultural transmission model in the simulation Sugarscape. Sugarscape is a complex simulation that models many different types of interactions between agents that roam their world in search of food. Each agent has a set of beliefs which can be adopted by the agents with which it interacts. In the simpler Axelrod Culture Model, agents have a set of beliefs, but stay in a fixed position and do not seek food. In this model, interactions with the nearest neighbors can lead an agent to adopt a belief of a neighbor. By turning off all but the cultural transmission interactions in Sugarscape, the group showed that Sugarscape behaves in a comparable way to the Axelrod Culture Model. This allowed the group to better verify that Sugarscape's cultural transmission mechanisms had been correctly implemented (Axtell, 1996). Since both Sugarscape and the Axelrod Culture Model implement the same model, this docking exercise verifies the implementation of the model in both simulations, but does not validate the precision of the model in describing the real world. Figure 1.2 illustrates this situation.

1.2 Interest in Understanding the Open Source Software Movement at Sourceforge

The Open Source Software group at Notre Dame is interested in better understanding the unique and successful open source software development model (Madey, 2004). To do so, the group studied Sourceforge.net, a website hosting the largest collection of open source software development in the world.

The development model for open source software is different from the model traditionally used in commercial software development. Unlike commercial development, the open source software model utilizes little centralized control over the software development process. There is no development cycle beginning with a formal analysis of the features the software needs. There is no centralized management controlling the development of the software nor is there a formal testing phase performed after development is completed. Instead, the software's code is made public for peer review, allowing any interested software developer to test, critique, and modify the software as they please. New modifications are made public, allowing for massive peer review. Advocates argue that this model has sped up the process of adding and debugging new software features to create robust, stable software quicker than is possible using the commercial software development model (Bollinger, 1999, 31).

The open source software model has a general appeal because it illustrates a paradigm shift from a centralized to decentralized mode of development. In his book, *The Cathedral and the Bazaar*, Eric Steven Raymond (Raymond, 2000) expresses his amazement that a paradigm that is so different from traditional intuitive understanding of software development works. Raymond had believed that simpler programs could be developed by decentralized means, but that more complex projects had to be developed using the traditional development model. He compares the traditional model to the process of constructing a cathedral, "I believed that the most important software (operating systems and really large tools

like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation.” In contrast, he compares the decentralized method exhibited in the Linux open source development community to a busy bazaar, out of which software emerges,

“No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches ... out of which a coherent and stable system could seemingly emerge only by a succession of miracles. The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders” (Raymond, 2000).

This movement has produced very successful software, including the Linux operating system, Apache, and the X11 graphical communication system, among others. Realizing the potential of this development model, Netscape adopted the open source development model for its popular web browser. Apple Computer also has adopted this model for the Unix kernel underlying its popular OS X operating system.

To study this development model, the Notre Dame group studied the website Sourceforge.net, which hosts the largest repository of open source software projects in the world. It currently hosts 78,000 open source projects and over 800,000

software developers. It provides them computer space for storing software, communication tools, an interface between the public and the developers, tools for merging modified code into the main code, and tools for releasing new versions of the software.

1.3 Explaining the Evolution of Sourceforge with Social Network Models

To better understand the dynamics of the open source movement, the group studied the evolution of the network at Sourceforge. The group investigated whether the social network formation model proposed by physicists Alberto-László Barabási and Réka Albert explained the evolution of the Sourceforge community. Barabási and Albert found that many networks occurring in different domains exhibit similar topological properties, leading them to suspect that similar generalized mechanisms were influencing the growth of these networks. Most generally, a network is a set of nodes connected by links. A network can model the relationships in a social group, for example, with nodes representing the people. The nodes are linked with one another if the people have spoken to each other. The World Wide Web can also be modeled as a network with nodes representing webpages and links occurring between nodes if one webpage has a hyperlink to the other. The degree of any node is the number of links connecting to it. Both of these networks have a distribution of degrees that follows a power law. Neither the network formation models of Paul Erdős and Alfred Rényi nor that of Duncan

Watts and Steven Strogatz model networks with such a degree distribution (Barabási, 1999, 173-5).

Barabási and Albert showed that networks with power-law distributions would form if nodes entering the network attached to existing nodes using the selection criteria of preferential attachment with fitness. Preferential attachment with fitness describes a stochastic process in which nodes entering a network are proportionately more likely to connect to existing nodes with a higher product of degree and fitness. The fitness value assigned to a node quantifies the value of that particular node to network. For example, in the network of websites, the search engine Google has more value to the average computer user than a teenager's personal webpage. Hence, an appropriately higher fitness would be assigned to Google over the teenager's website. Both closed form math and computer simulations have verified that networks obeying the preferential attachment and fitness model develop a topology with a power-law degree distribution (Barabási, 1999, 178-81).

1.4 Empirical Evidence of Power Law at Sourceforge

With empirical data showing a power law degree distribution existing at Sourceforge, the Notre Dame group implemented 4 models of Sourceforge to test whether the Barabási-Albert model of preferential attachment and fitness could explain the empirical evidence.

Empirical data from Sourceforge showed two social networks existing with a power law degree distribution, indicating that the network evolution mechanisms theorized by Barabási and Albert may be influencing the formation of this community. One social network, the “developer-developer” network, links developers with each other if they work on a common project. The second, the “project-project” network, links projects to each other if they share a common developer. With data obtained from Sourceforge, the group found that the developer-developer and project-project networks have degree distributions which follows a power law (Madey, 2002).

1.5 Implementation of Sourceforge Simulations Using the Barabási-Albert Model of Network Growth

To investigate whether the Barabási-Albert theory could explain this observation, the group implemented four agent-based simulations using the simulation package SWARM. During each time step, a new developer stochastically chooses either to join a project or to create a new project. Existing developers stochastically choose either to join a project, to abandon a current project, to create a new project, or to make no change. The four variations each differ in the criteria by which a developer selects a particular project to join. The ER model, based on the Erdős-Rényi model of network formation, serves as a baseline, with developers equally likely to join any project. In the Barabási Albert, “BA,” model, developers are more likely to join projects with higher degrees. In

the “BA with constant fitness” model, each project is assigned a fitness, with developers more likely to join projects with higher products of fitness and degree. The “BA with dynamic fitness” model is similar to the constant fitness model, except the fitness decreases periodically. As the programs run, it archives data on the projects that developers have joined. A database script analyzes this data to determine the degree distributions of the developer-developer and project-project networks.

2.0 PROCEDURE

2.1 Simulation Rewrite in RePast

We migrated these simulations from SWARM to RePast to take advantage of the clean JAVA implementation of RePast. RePast offers a much cleaner programming environment than SWARM (Xu, 2003).

2.2 Docking Process

We used both visual observations of graphs and a chi-squared test to determine whether results aligned. To align, it must be likely that the resulting distributions from corresponding implementations come from the same underlying distribution. The results studied were the number of developers and projects vs. time and the periodic developer-developer and project-project degree distributions. Graphs provided a visual means of determining whether two distributions could come from same underlying distribution (Xu, 2003). A chi-squared test determined whether the average of SWARM distributions tracks with the average of RePast distributions. We ran the chi-squared test to determine whether the following null hypothesis should be rejected.

H_0 : the RePast distribution conforms with the SWARM distribution

H_1 : the RePast distribution does not conform with the SWARM distribution

Given

x_s , average of a SWARM distribution

x_R , average of a RePast distribution

the group calculated chi-squared

$$\chi_0^2 = \sum_{\substack{\text{all} \\ \text{dist.}}} \frac{(x_s - x_R)^2}{\frac{(x_s + x_R)}{2}}$$

with a significance level of .05.

We used these comparisons to find and correct errors in the SWARM and RePast implementations. Discrepancies were analyzed to determine possible causes. This process found a mistake in the coding of a database command. Also, the RePast implementations began at time step 1, whereas the SWARM simulation began at time step 0. We also discovered the random number seed was not correctly passed to the random number generator in the SWARM simulations. To eliminate potential discrepancies arising from the different random number generators used in SWARM and RePast, the SWARM implementations were modified to use the same generator used in the RePast model, the Colt Uniform generator based on the Mersenne Twister algorithm.

3.0 RESULTS

The comparison found good alignment between the two implementations of the BA model and the BA with constant fitness model. Visual inspection showed the number of developers vs. time, number of projects vs. time, and degree distribution results to align well. The chi-squared test did not reject the null hypothesis for any result. For the ER and BA with dynamic fitness implementations, the number of developers vs. time and the number of projects vs. time partially align with each other, with a divergence after month 11 in the number of projects for the dynamic fitness model. The chi-squared test does not reject the null hypothesis number of developers vs. time for ER model. The number of projects vs. time barely passes, however. In the BA with dynamic fitness implementation, the chi-squared test rejects the null hypothesis on the number of projects vs. time. The chi-squared test does not reject any null hypothesis for the developer-developer or project-project degree distributions (see Appendix).

Using another test, the group calculated the confidence that corresponding distributions came from the same distribution. The test determined the possible range of differences between the mean of the two distributions. If this range included 0, then it would be possible that the two distributions came from the same underlying distribution.

$$\bar{X}_S \square \bar{X}_R \pm t_{\square/2, \square} s.e.(\bar{X}_S \square \bar{X}_R)$$

where

$$s.e.(\bar{X}_S \square \bar{X}_R) = \sqrt{\frac{S_1^2}{R_1} + \frac{S_2^2}{R_2}}$$

S_i : variance

R_i : # runs

for uncorrelated runs.

For the ER simulation and BA simulation almost all of the intervals contain 0.

As time increased, however, the range became larger because of larger variances within the distributions. For the BA with constant fitness simulation, the number of developers vs. time begins with a strong bias towards the SWARM implementation, indicating that the SWARM implementation has more developers than the corresponding RePast implementation during the beginning months.

Also, in the BA with dynamic fitness model, the number of projects vs. time interval calculation shows that at later months the RePast implementation has more projects than the SWARM implementation.

4.0 DISCUSSION

4.1 Discussion of Results

We can increase our confidence that BA models and BA with constant fitness models were correctly implemented in SWARM and RePast because the results do not fail visual and statistical tests run on data. We know that substantial differences still exist between SWARM and RePast implementations of the ER and BA with dynamic fitness models.

The failures occurred in the most complex behaviors of the simulation, as expected. All implementations aligned on the number of developers vs. time, a result fixed by user parameters. Differences between implementations occurred in the number of projects vs. time, a result that depends both on the number of developers in the system and on the behaviors that the developers chose. Differences also occurred in the project-project degree distributions which is influenced by the number of projects, number of developers, and the topology of connections formed between developers and projects.

In the ER model, the number of projects vs. time was almost rejected. This poor alignment may be one cause for the rejection of the project-project degree distribution.

It is surprising, however, that in the BA with dynamic fitness model, the number of projects vs. time do not align, but the project-project degree distribution, which is influenced by the number of projects, does. Future work will investigate this.

In addition, the confidence interval calculations showed that ER implementation and the BA implementation could come from the same underlying distribution. This test showed differences in the number of developers vs. time in the BA with constant fitness implementation and in the number of projects vs. time in the BA with dynamic fitness implementation. The problem with the developers in the BA with constant fitness implementation is surprising, especially since it occurs during the first few months of the simulation. Investigation is needed. Improvement in the resolution of all of these calculations can be improved by running more simulations or by correlating the random number sequences used by corresponding simulations.

4.2 Limitations of this Docking Exercise

This docking exercise is not a perfect or complete example of docking. In programming the RePast implementations, the SWARM code was analyzed. Since the RePast models were not developed independently of the SWARM models, errors in the original implementations may have been included in the RePast rewrite, but would not be found in this docking exercise. To prevent this, the RePast models should have been developed independently of the SWARM implementations, directly from a model description. In addition, we originally intended to use the diameter of the networks as another simulation result on which to compare corresponding implementations. Unfortunately, we were unable to develop an efficient algorithm to calculate the diameters of the networks. Finally,

we need to run more iterations of each implementation to increase their confidence in the docking.

4.3 Lessons from this Docking Exercise

Especially with the migration of one model from one simulation package to another, it is relatively easy to determine whether results align. A similar docking exercise on a model migration would cost scientists little in terms of time but potentially find important errors in implementation. Additionally, through a docking exercise, scientists may discover errors in the original implementation of a program. Finally, people unfamiliar with the simulations became familiar with them because of the exercise.

4.4 Future Plans for the Simulations

We plan to investigate the alignment error in the BA with dynamic fitness implementation in order to achieve a complete docking and to run more iterations of each implementation. In addition, we may attempt a correlated comparison of the simulations. Corresponding events in the SWARM and RePast simulations will each use the same random number. Doing so will increase the resolution in our confidence interval calculation to help better determine whether the two distributions come from the same underlying distribution.

After completing the docking experiment, we will begin to use the RePast implementations to investigate the development of Sourceforge.

5.0 CONCLUSION

In general, scientists can use a docking process to increase their confidence in the implementations of a model. This method can find subtle errors in the implementations which often would not be found using other methods of verification. In addition, scientists can improve their confidence in the docking by aligning implementations on multiple results. As in the results presented, errors not manifested in one result may be visible in other results.

WORKS CITED

- Axelrod, Robert. "The Convergence and Stability of Cultures: Local Convergence and Global Polarization," Santa Fe Institute working paper 95-03-028.
- Axtell, R., R. Axelrod, J.M. Epstein, and M.D. Cohen. "Aligning Simulation Models: A Case Study and Results." *Computational and Mathematical Organization Theory*. 1 (1996): 123-41 <http://www-personal.umich.edu/~axe/research/Aligning_Sim.pdf>.
- Banks, Jerry, John S. Carson II, Barry L. Nelson, and David M. Nichol. *Discrete-Event System Simulation*. third ed. Upper Saddle River: Prentice, 2001.
- Barabási, Albert-László. *Linked: The New Science of Networks*. Cambridge: Perseus, 2002.
- Barabási, Albert-László and Réka Albert. "Emergence of Scaling in Random Networks." *Science* 286 (1999): 509-12.
- Barahona, Jesus M. Gonzalez, Pedro de las Heras Quiros, and Terry Bollinger. "A Brief History of Free Software and Open Source." *IEEE Software*. 16.1 (1999): 32-3
<<http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16092&puNumber=52>>.
- Bollinger, Terry and Peter Beckman. "Linux on the Move." *IEEE Software*. 16.1 (1999): 30-5
<<http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16092&puNumber=52>>.
- Hecker, Frank. "Setting Up Shop: The Business of Open-Source Software." *IEEE Software*. 16.1 (1999): 45-51

<http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=16092&puNumber=52>>.

Madey, Greg. *Research Project on the Free/Open Source Software (F/OSS)*

Development Phenomenon. University of Notre Dame. 5 April 2004

<http://www.nd.edu/~oss>>.

Madey, Greg, Vincent Freeh, and Renee Tyron. "Understanding OSS as a Self-Organizing Process." The 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering. (ICSE 2002). Orlando, FL, 2002.

Raymond, Eric Steven. *The Cathedral and the Bazaar*. 11 September 2000

<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html>>.

Wu, Ming-Wei and Ying-Dar Lin. "Open Source Software Development: An Overview." *Computer*. 34.6 (2001): 33-8

<http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=20074&puNumber=2>>.

Xu, Jin, Yongqin Gao, Jeffrey Goett, and Gregory Madey. "A Multi-Model Docking Experiment of Dynamic Social Network Simulations." *Agents2003*, Chicago, IL, October 2003

http://www.nd.edu/~oss/Papers/Agents2003_Docking.pdf>.

Appendix

Model name-result	chi-squared statistic	critical value	reject?
ER			
developer count vs. time	1	33	
project count vs. time	28	33	
BA			
developer count vs. time	6	33	
project count vs. time	4	33	
BA with constant fitness			
developer count vs. time	6	33	
project count vs. time	12	33	
BA with dynamic fitness			
developer count vs. time	1	33	
project count vs. time	318	33	reject

ER Model

Developer-Developer Degree Distribution		
chi-squared statistic	critical value	reject?
0.06	9.49	
0.16	12.59	
0.36	14.07	
0.49	19.68	
1.52	22.36	
1.93	28.87	
3.22	31.41	
1.85	33.92	
2.48	37.65	
2.68	44.99	
3.24	48.60	
4.19	53.38	
5.98	59.30	
5.20	66.34	
7.65	72.15	
8.13	81.38	
9.78	90.53	
11.11	100.75	
12.33	107.52	
9.35	117.63	
14.15	128.80	
17.65	136.59	
14.64	147.67	
16.06	150.99	

Project-Project Degree Distribution		
chi-squared statistic	critical value	reject?
40.93	37.65	reject
60.08	48.60	reject
70.04	53.38	reject
70.59	55.76	reject
69.16	54.57	reject
70.19	53.38	reject
78.10	53.38	reject
80.38	51.00	reject
75.34	46.19	reject
83.30	43.77	reject
80.22	38.89	reject
69.24	35.17	reject
56.12	33.92	reject
42.05	32.67	reject
37.85	30.14	reject
38.91	26.30	reject
33.49	25.00	reject
30.90	25.00	reject
27.91	25.00	reject
23.22	23.68	
20.71	22.36	
15.17	21.03	
11.88	21.03	
11.52	22.36	

BA Model

Developer-Developer Degree Distribution		
chi-squared statistic	critical value	reject?
1.06	5.99	
0.34	7.81	
0.54	9.49	
0.26	9.49	
0.43	11.07	
0.38	11.07	
0.36	14.07	
0.33	14.07	
0.42	12.59	
0.61	14.07	
0.63	14.07	
0.63	15.51	
0.43	14.07	
0.51	15.51	
0.35	15.51	
0.46	16.92	
0.67	15.51	
0.24	15.51	
0.60	16.92	
1.12	16.92	
0.47	16.92	
0.61	16.92	
1.17	18.31	
0.60	18.31	

Project-Project Degree Distribution		
chi-squared statistic	critical value	reject?
1.15	36.42	
2.20	52.19	
2.38	55.76	
2.03	58.12	
2.38	64.00	
2.30	65.17	
2.07	62.83	
1.50	62.83	
1.66	61.66	
4.45	62.83	
4.01	65.17	
2.97	62.83	
3.11	60.48	
2.79	60.48	
3.21	58.12	
2.90	55.76	
1.85	56.94	
1.90	53.38	
2.77	52.19	
1.92	49.80	
1.82	51.00	
1.22	47.40	
2.72	47.40	
1.20	44.99	

BA with Constant Fitness Model

Developer-Developer Degree Distribution		
chi-squared statistic	critical value	reject?
1.15	7.81	
0.52	7.81	
0.35	9.49	
0.71	11.07	
0.85	12.59	
0.74	11.07	
1.01	14.07	
1.49	14.07	
0.60	14.07	
0.86	15.51	
0.46	15.51	
1.02	16.92	
1.07	16.92	
1.45	16.92	
1.65	16.92	
0.94	16.92	
0.70	18.31	
0.57	18.31	
1.88	16.92	
0.86	16.92	
1.02	16.92	
1.20	18.31	
0.89	18.31	
0.99	18.31	

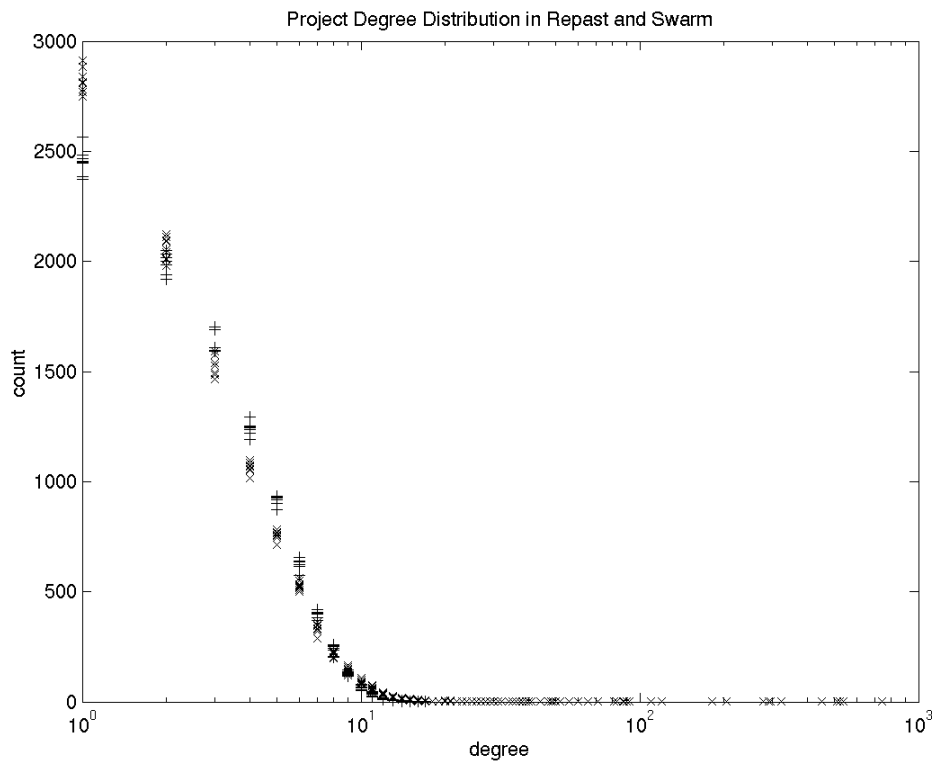
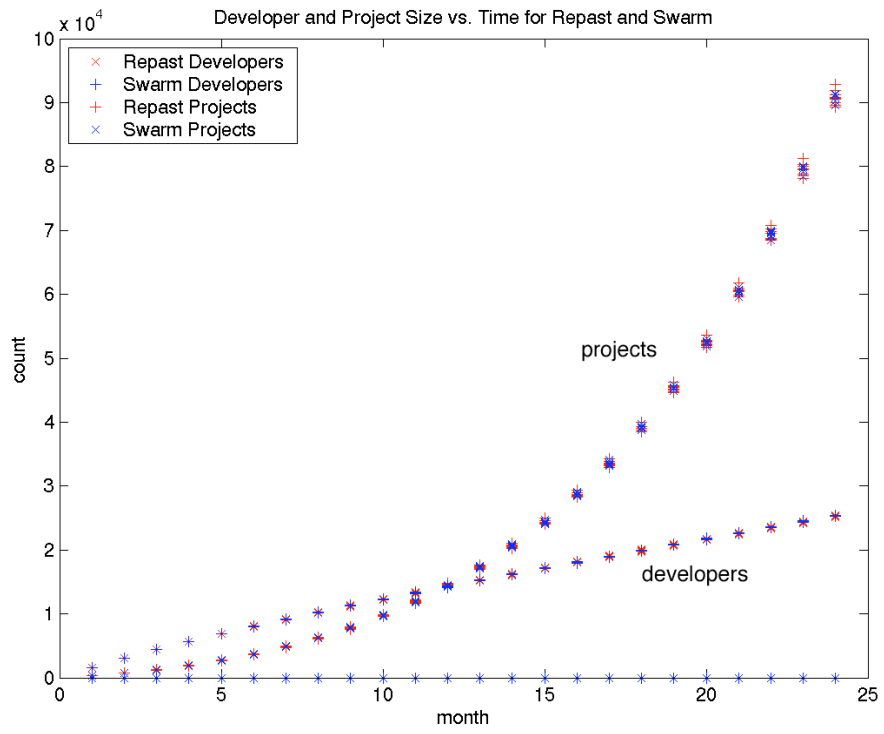
Project-Project Degree Distribution		
chi-squared statistic	critical value	reject?
1.07	35.17	
3.27	47.40	
2.08	48.60	
2.77	54.57	
5.25	60.48	
4.81	64.00	
5.22	66.34	
6.22	66.34	
5.62	67.50	
6.26	67.50	
5.67	69.83	
5.42	67.50	
4.66	66.34	
6.19	68.67	
4.27	64.00	
5.26	66.34	
6.09	66.34	
6.37	62.83	
4.61	61.66	
5.18	64.00	
5.10	59.30	
5.82	61.66	
5.19	58.12	
4.60	55.76	

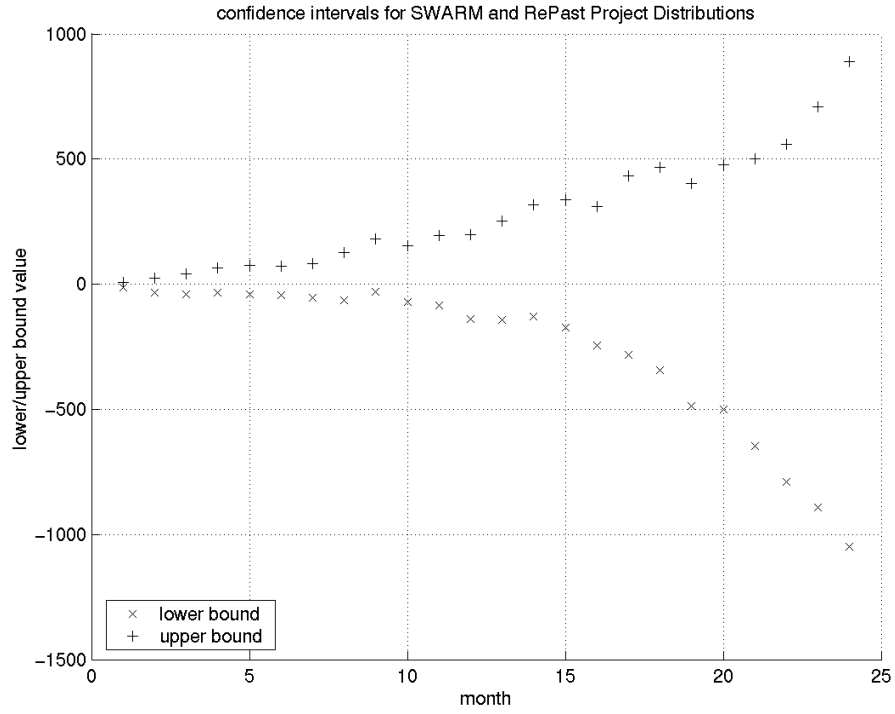
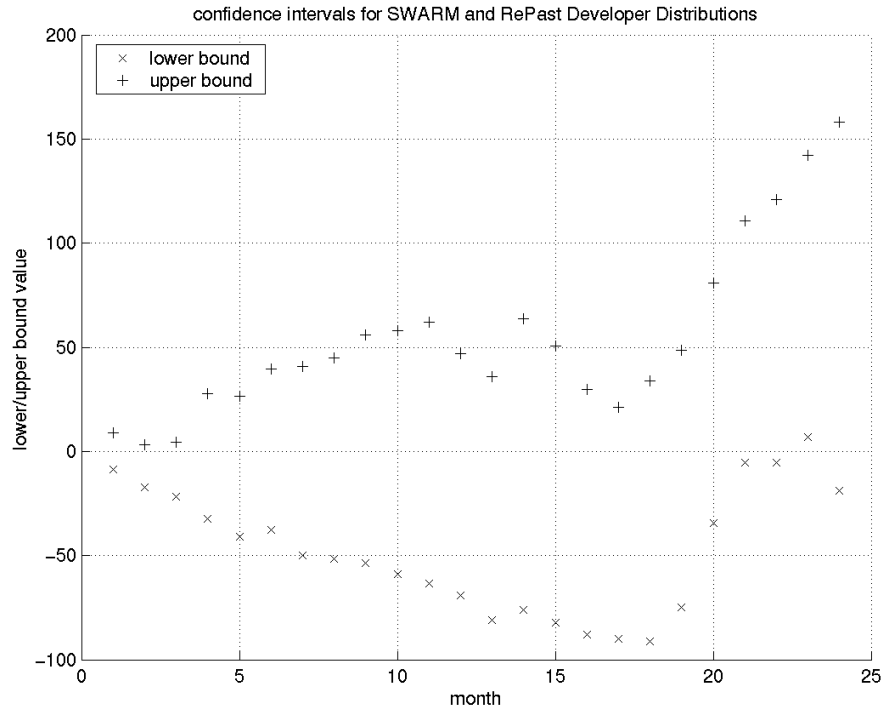
BA with Dynamic Fitness Model

Developer-Developer Degree Distribution		
chi-squared statistic	critical value	reject?
0.83	5.99	
0.50	7.81	
0.43	9.49	
0.18	9.49	
0.33	11.07	
0.24	11.07	
0.48	14.07	
0.27	14.07	
0.18	12.59	
0.37	14.07	
0.67	14.07	
0.64	15.51	
0.33	14.07	
0.63	15.51	
0.26	15.51	
0.53	16.92	
0.89	15.51	
0.19	15.51	
0.89	16.92	
1.48	16.92	
0.52	16.92	
0.69	16.92	
1.24	18.31	
0.65	18.31	

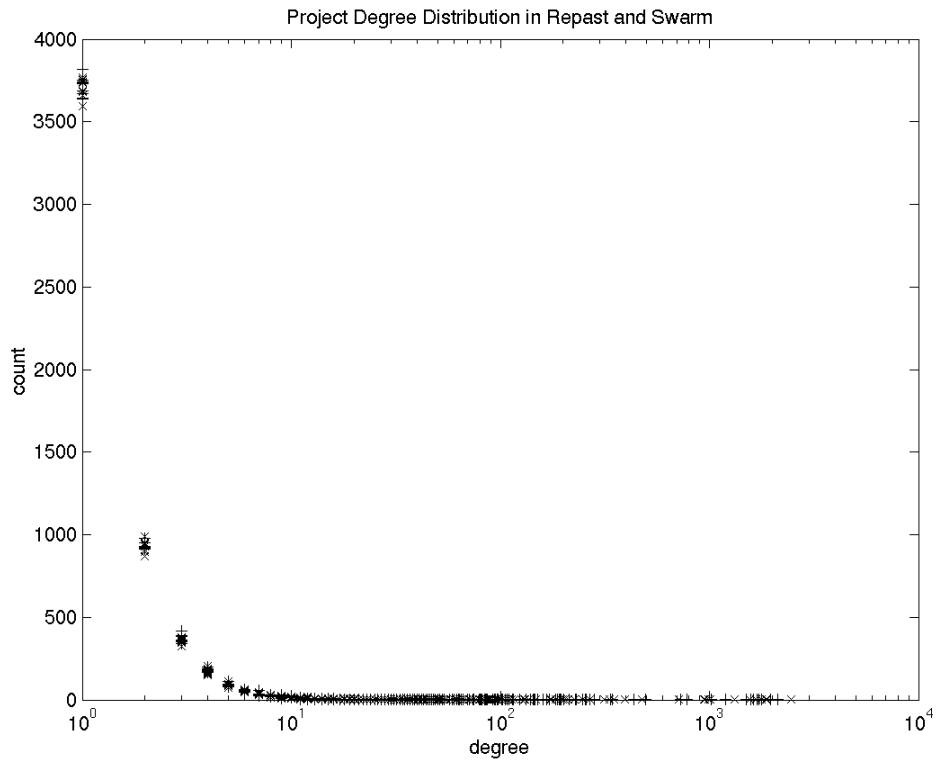
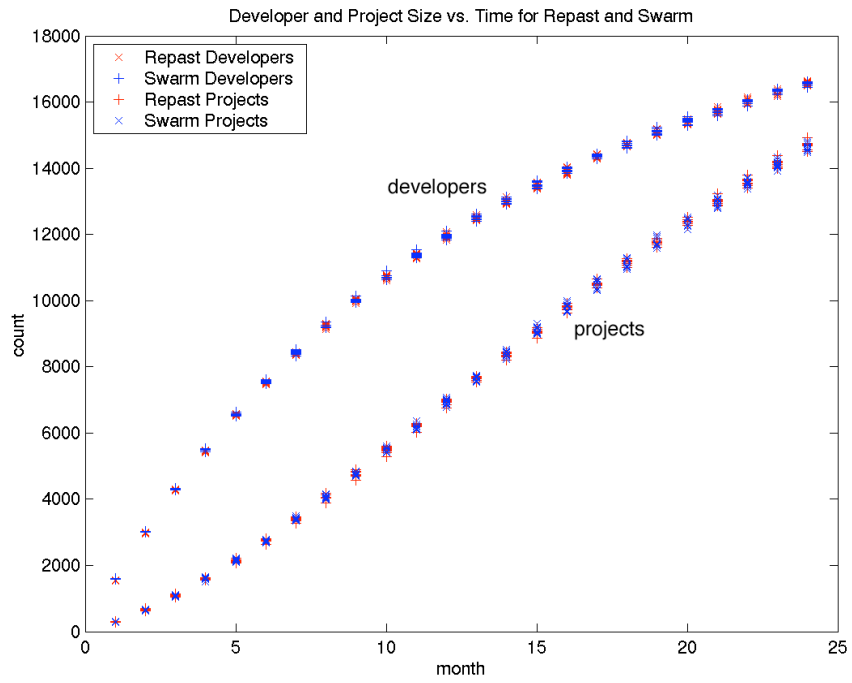
Project-Project Degree Distribution		
chi-squared statistic	critical value	reject?
1.70	37.65	
1.77	46.19	
5.31	53.38	
5.71	62.83	
5.50	60.48	
5.64	58.12	
5.34	62.83	
7.41	67.50	
8.60	65.17	
13.46	70.99	
17.69	69.83	
21.74	68.67	
24.41	64.00	
28.90	64.00	
28.80	65.17	
33.44	66.34	
40.49	72.15	
44.10	70.99	
48.14	75.62	
50.11	74.47	
57.50	74.47	
61.30	74.47	
60.24	76.78	
69.40	75.62	

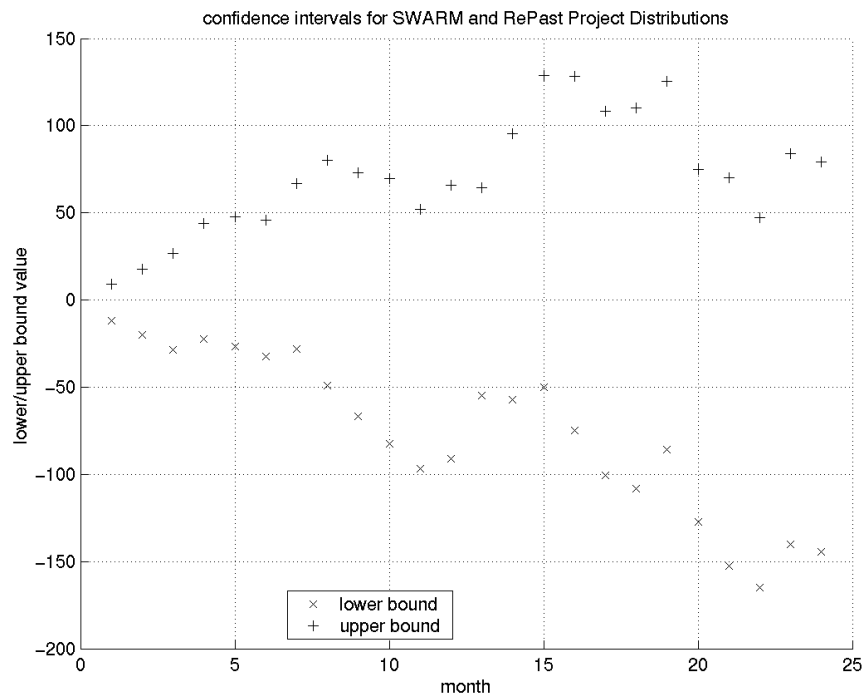
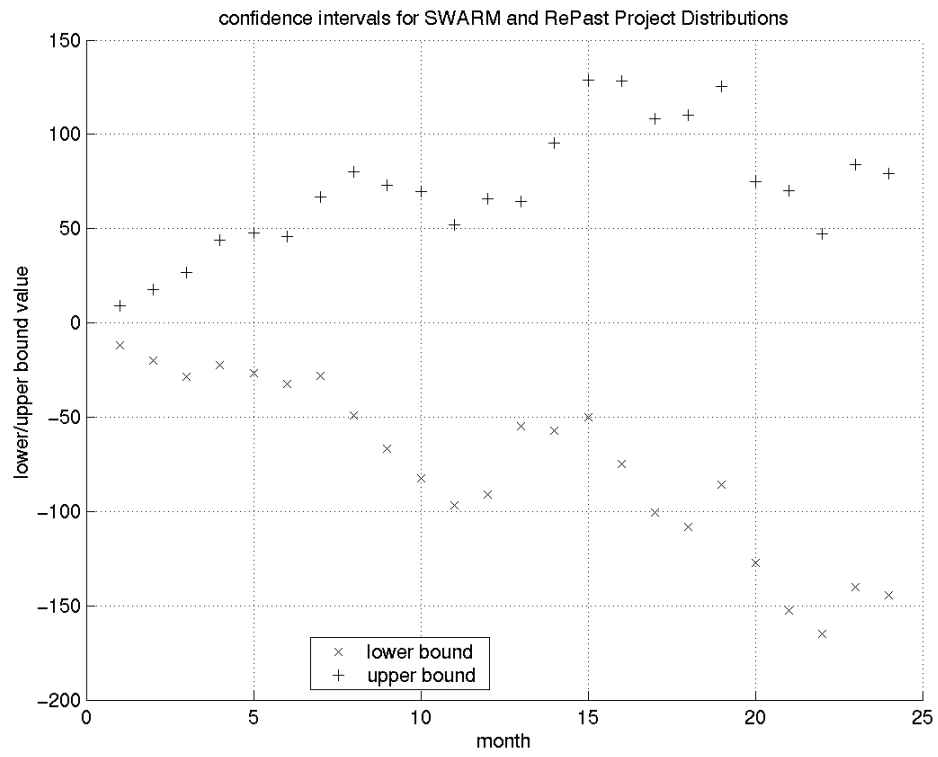
E.R. Model

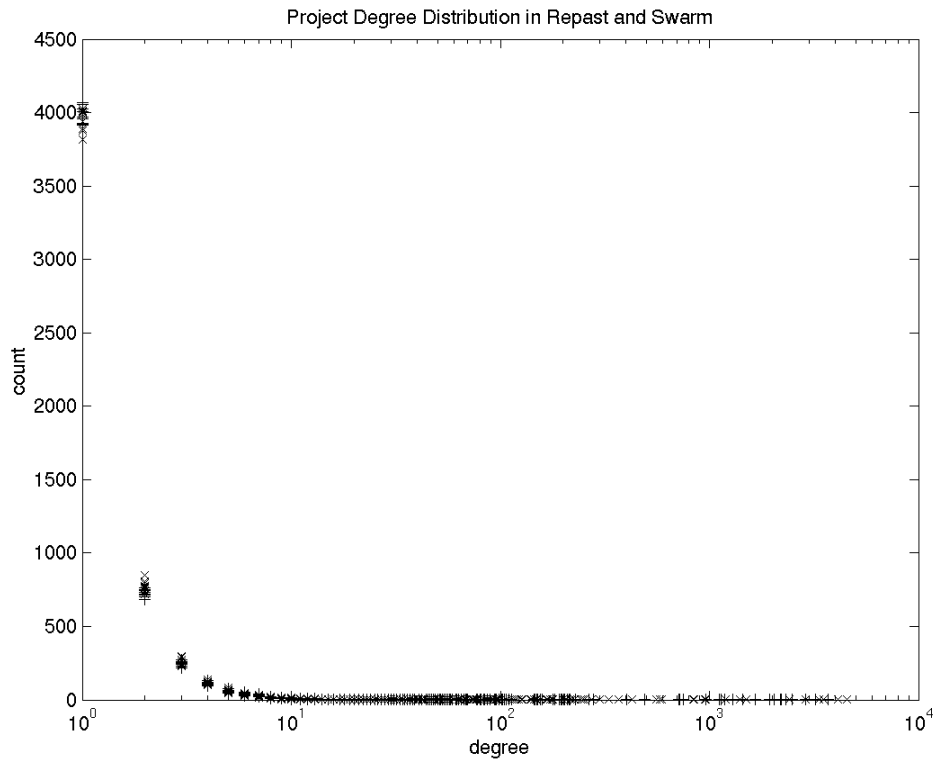
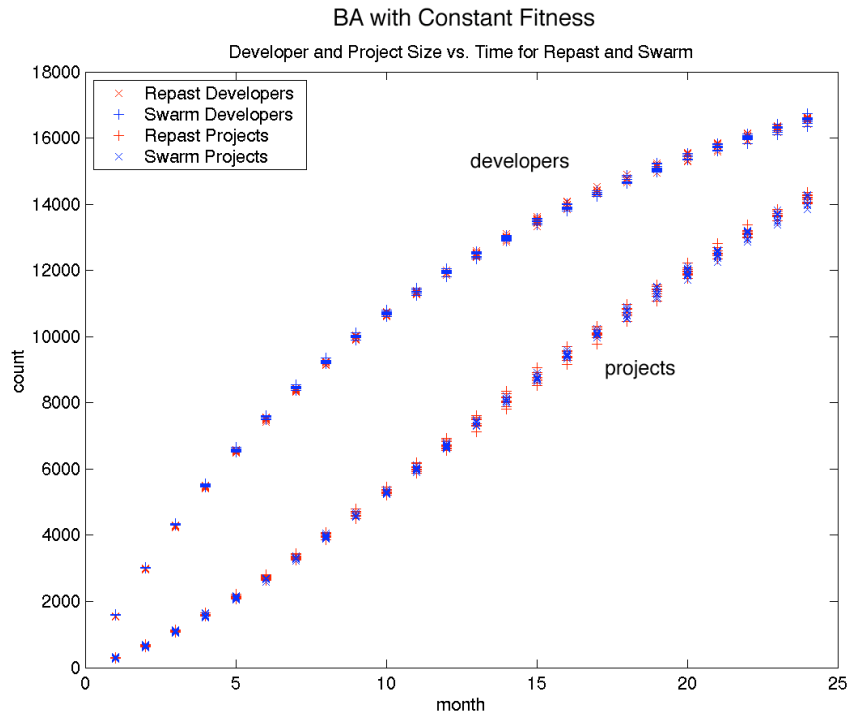


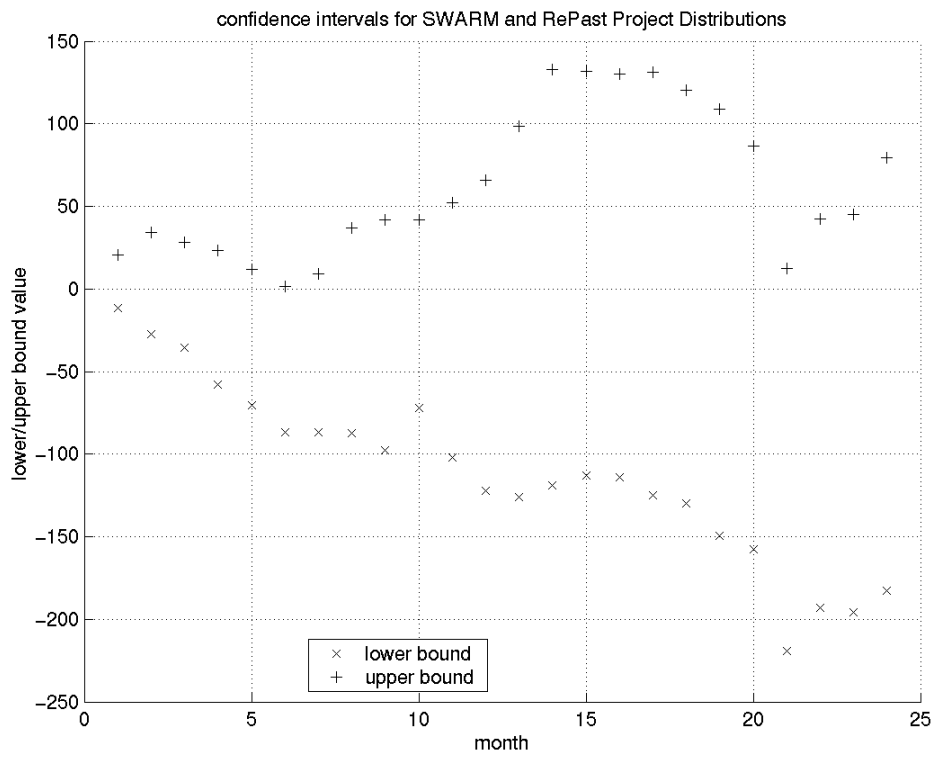
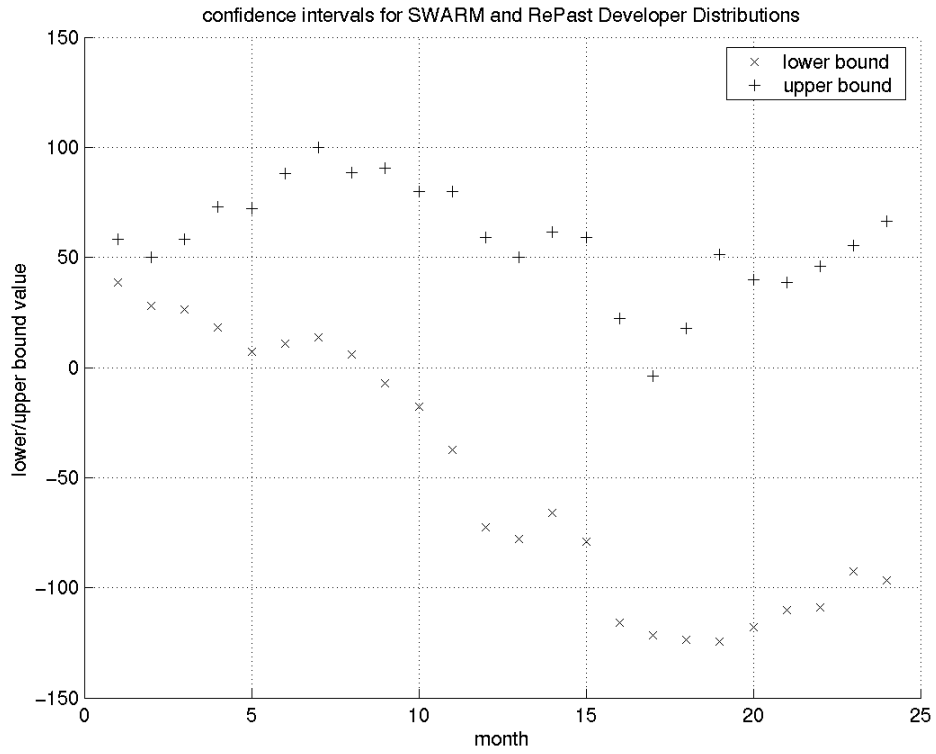


BA Model



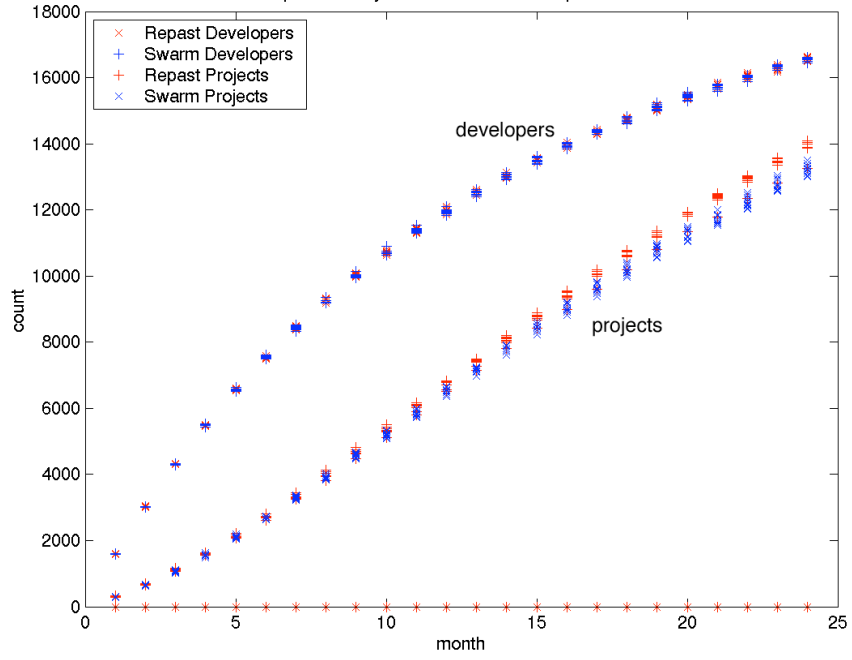






BA with Dynamic Fitness

Developer and Project Size vs. Time for Repast and Swarm



Project Degree Distribution in Repast and Swarm

