What Leads to Innovation?
An Analysis of Collaborative Problem-Solving

Randy M. Casstevens
Department of Computational Social Science
George Mason University
Fairfax, VA 22030
rcasstev@gmu.edu

January 31, 2011

Abstract

Collaboration within large groups of individuals is difficult to analyze due to the large number of interactions involved. This paper explores the problem-solving process of software developers with three types of analysis. First, regression was used to investigate the effect of two problem characteristics, size and modularity, on the innovation process. Second, the distributions of developers per project and projects per developer was calculated to establish if they have heavy tailed distributions like seen in the open source community. This will examine whether heavy tailed distributions can form over a relatively short time period, approximately one week. Finally, the programming contests were examined to look for evidence of creative destruction.

1 Introduction

'Innovation' can be viewed as any improvement over an existing solution to a problem. Therefore, a greater understanding of the problem-solving process may increase either the rate or effectiveness of innovation. Deriving a clearer picture of how large groups of individuals contribute to a problem-solving task is difficult. This is due to the large number of interactions between group members. This problem is often compounded because distinct innovations are difficult to compare. Studying problem solving within software development is the ideal domain because the progression of computer code permits examination in detail and can be self-documenting. Also, metrics for solution comparison tend to be easier to quantify (e.g. solution quality or execution time).

Data from MATLAB programming contests were gathered and analyzed [1]. This contest data has shown to be an excellent way to explore collaborative

problem-solving. The MATLAB programming contests are different from traditional contests in that a submission is not necessarily written by a single individual. In order for a program to be scored, the software developer must submit it to the contest administrators. Once submitted, the program is scored and made available to all contest participants. Any participant can use previously submitted programs as the starting point for their submission, thus allowing for collaboration within a competitive contest. Moreover, these contests have an interesting mix of collaboration and competition reminiscent to the open source software movement [2].

Currently, the programming contest data has been analyzed three ways. First, linear regression was used to explore which characteristics of the problem being solved are the best predictors of innovation. With a better understanding of how the characteristics of the problem being solved influences the rate of innovation, obstacles like Brooks's Law may be avoided. Brooks's Law states that adding more software developers to a late project will make it later [3].

The second type of analysis examined the distributions of developers per project and projects per developer to investigate if they follow a heavy tailed distribution and possibility a power law. Power laws have been found for the developers per project and project per developer distributions in the open source community [4]. The programming contest data provided an opportunity to examine if this type of pattern could emerge in programming contests that occur over a relatively short time period, for approximately one week. Moreover, finding these types of distributions in the programming contest data may help draw parallels between the relatively simple programming contests and the highly complex community of open source software developers. Studying a simpler system that resembles the open source community may shed light onto previously unknown aspects of open source software development.

In the last analysis, changes in the submission population was recorded to examine if there is evidence of creative destruction. In creative destruction, the introduction of a new innovation causes other products to no longer be used and was credited by Schumpeter as the "fundamental impulse that sets and keeps the capitalist engine in motion" [5, pg. 83]. By calculating the diversity of the submission population before and after a best-so-far submission, the reduction in submission diversity could be investigated and the narrowing of the problem-solvers focus could be measured. For comparison, the same operation was also performed for the submissions that were not best-so-far submissions. A statistical analysis was performed to see if the better submissions caused narrowing of focus by the problem-solvers and thereby, creative destruction.

2 Problem Characteristics Regression Analysis

The regression analysis focuses on the influence of problem size and problem modularity on software developers ability to improve contest solutions. Data from twenty programming contests have been analyzed. There are two hypotheses being tested. First, problems that are more modular will have more

innovations by a larger number of contributors. Second, larger problems will have more innovations by a larger number of contributors. The analysis provides a quantitative way of comparing the importance of problem size and problem modularity on the innovation process. The analysis was done with ordinary least squares (OLS) regression on the following dependent and independent variables.

2.1 Dependent Variables

The dependent variables are the number of innovations per participant and the percentage of participants that were innovators. An innovation is simply anything that improved upon the best solution so far and an innovator is anyone that submitted one of those best-so-far solutions.

2.2 Independent Variables

The independent variables are the modularity and size of the problem being solved by the participants. The problem's modularity and size are not characteristics that are directly observable. Therefore, the participants' submissions were used to calculate an estimate of the modularity and size of the problem. Modularity was measured by the average number of sub-functions of all the submissions. Size was measured by the average number of nodes in the parse tree of all of the submissions. The size of the parse tree is a better indicator of program length than the number of lines of code. Number of lines of code as a measure of program length can be misleading due to differences in programming style that do not affect the execution of the program. Only submissions that did not generate an error during execution and received a score where used in these calculations.

One argument against using modularization as a predictor of innovation is the notion that more modules simply means longer programs which is the real reason for the increased amount of innovation. With this analysis, we can determine which factor explains that most variation in the variables measuring innovation.

The results discussed here used the average modularity and size for all of the contest submissions. Separate analyses were also done for the problem characteristics of the winning submissions and the average of the best-so-far submissions. These other variables were highly correlated with the ones reported here and did not change the nature of the results, so they were not included.

2.3 Excluded Data

Currently, the MATLAB programming contest website contains information about twenty-two programming contests. Two of the contests were left out of the analysis because they used considerably different scoring metrics. One was excluded because it was a visualization contest and thereby, the submissions do not have a objective way of being scored and compared. The other excluded contest scored submissions based on the number of characters in the

submitted program that correctly converted the input into the desired output. Unlike the rest of the contests, solution quality and execution time were not a component of a submission's score. Because this was a substantially different scoring method from the other contests it was excluded from the analysis. The data from the remaining twenty MATLAB programming contests were analyzed using regression.

2.4 Regression Results

Table 1 displays the results of the regression analysis. The first two columns used innovations per participant and the last two columns used percent of participants that were innovators as the dependent variable. The first and third rows used the modularity variable, average number of functions, as the independent variable, while the second and fourth rows used the size variable, average node count of the parse trees. The modularity variable was a significant predictor for both innovation variables, while the size variable was only significant for one. The size variable was not significant for the percentage of innovators variable and provides support that problem modularization is more important than problem size for promoting involvement by larger numbers of innovators. Furthermore, the modularity variables had a larger effect size (measured by R-squared) than the parse tree size variables.

3 Distributions of Developers and Projects

In open source software, it has been discovered that the developers per project and projects per developer follow a power law distribution [4]. For this paper, a similar analysis was performed for the programming contests. Determining the developers was straightforward, but establishing 'projects' within a contest was not quite as clear. During the contest, the participants were given the opportunity of specifying another submission as the basis of their code. However, this information was incomplete, as this information was not always given. Therefore, 'projects' were established directly from the submissions themselves.

In order to group the similar submissions to make a 'project', the parse tree of each submission was generated and each function was compared with all other functions submitted during the contest. User defined names of functions and variables were not used during the comparison because those are easily changed to obfuscate the code without affecting the execution of the code. Also, the order of the branches in the parse tree were not used in the comparison, since those changes may or may not affect the execution of the code. Furthermore, by not being concerned with the order of the branches in the parse tree, it helped reduce the execution time of this already long running process. The comparison between two functions was between zero and one and measured the proportion of parse tree nodes that were in both functions. These function comparisons allowed the source code to be converted into a set of function types. A unique set of function types were considered a 'project'. Due to the large execution

Independent Variables:	Dependent Variable: Innovations per Participant		Percent Innovators	
Average Number Functions				
Coefficient	**0.089**			
p-value	**0.002**			
Average Node Count				
Coefficient		**1.9E-04**		
p-value		**0.010**		
Average Number Functions				
Coefficient			**0.007**	
p-value			**0.044**	
Average Node Count				
Coefficient				0.2E-04
p-value				0.061
Intercept	0.802	0.874	0.198	0.200
R-squared	0.431	0.315	0.207	0.181
Adjusted R-squared	0.399	0.277	0.163	0.136
F-statistic	13.630	8.281	4.705	3.987

Note: Values in **bold** are significant at the 5% level.

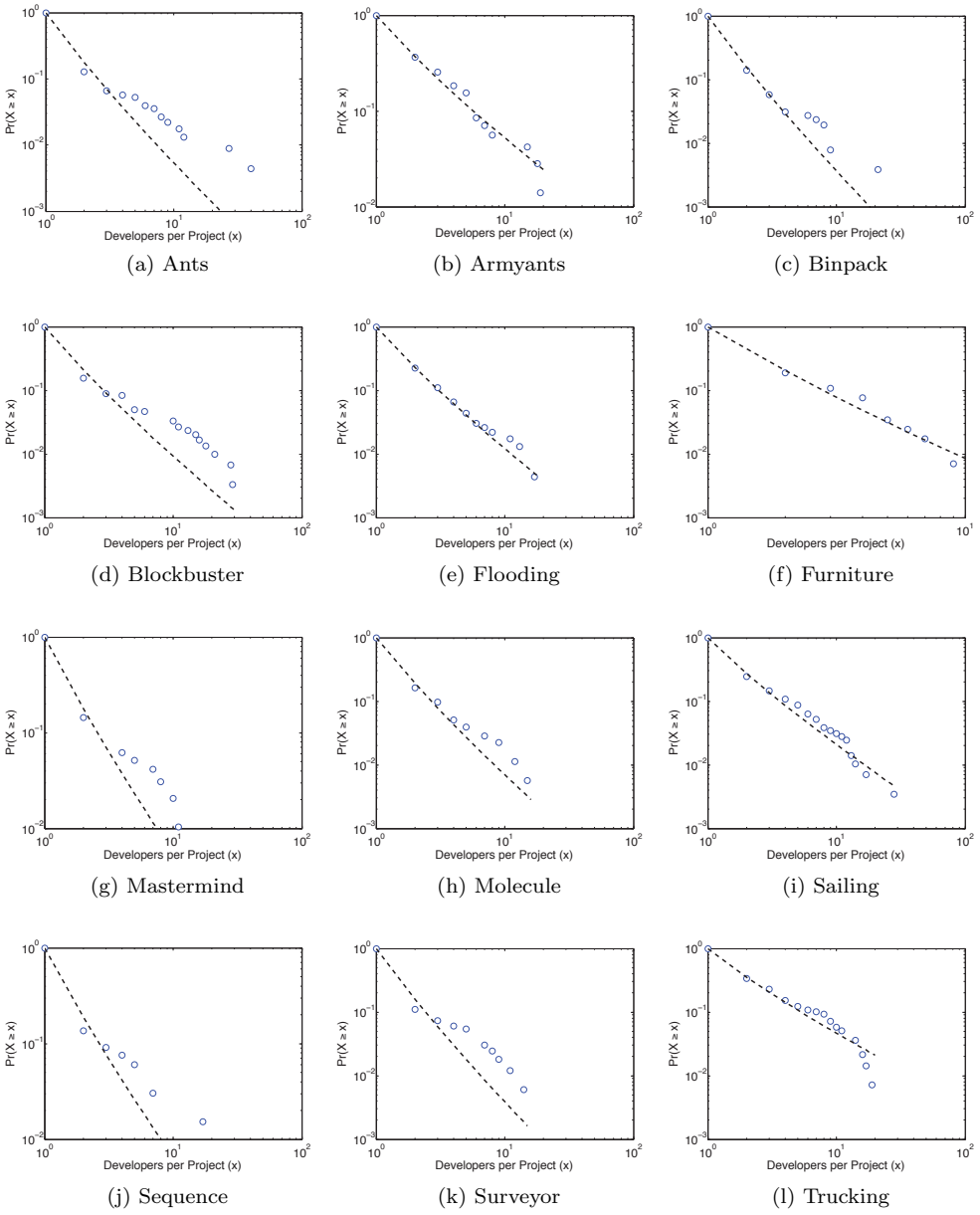
Table 1: Regression results from the MATLAB contests with innovations per participant and percentage of participants that were innovators as the dependent variables.

time and memory requirements of calculating the function types, only twelve of the twenty programming contests with smaller number of submissions were used for this analysis.

Figure 1 shows the developers per project and Figure 2 projects per developer. All of these plots are on a log-log scale. Most of the distributions display a heavy tail. Furthermore, five of the developer per project distributions and eight of the projects per developer distributions were not ruled out as being a power law (see Table 2 and 3) using the method described in Clauset et al. [6]. This gives some indication that a power law distribution can emerge in a relatively short time period and with a modest number of observations.

4 Evidence of Creative Destruction

Schumpeter's theory of creative destruction describes how existing technologies can be replaced by a new technology. The programming contest data provides a microcosm to study the "the perennial gale of creative destruction" [5]. There are new approaches to the problem being introduced throughout the contest. If creative destruction is present in the data, then it would be expected that when



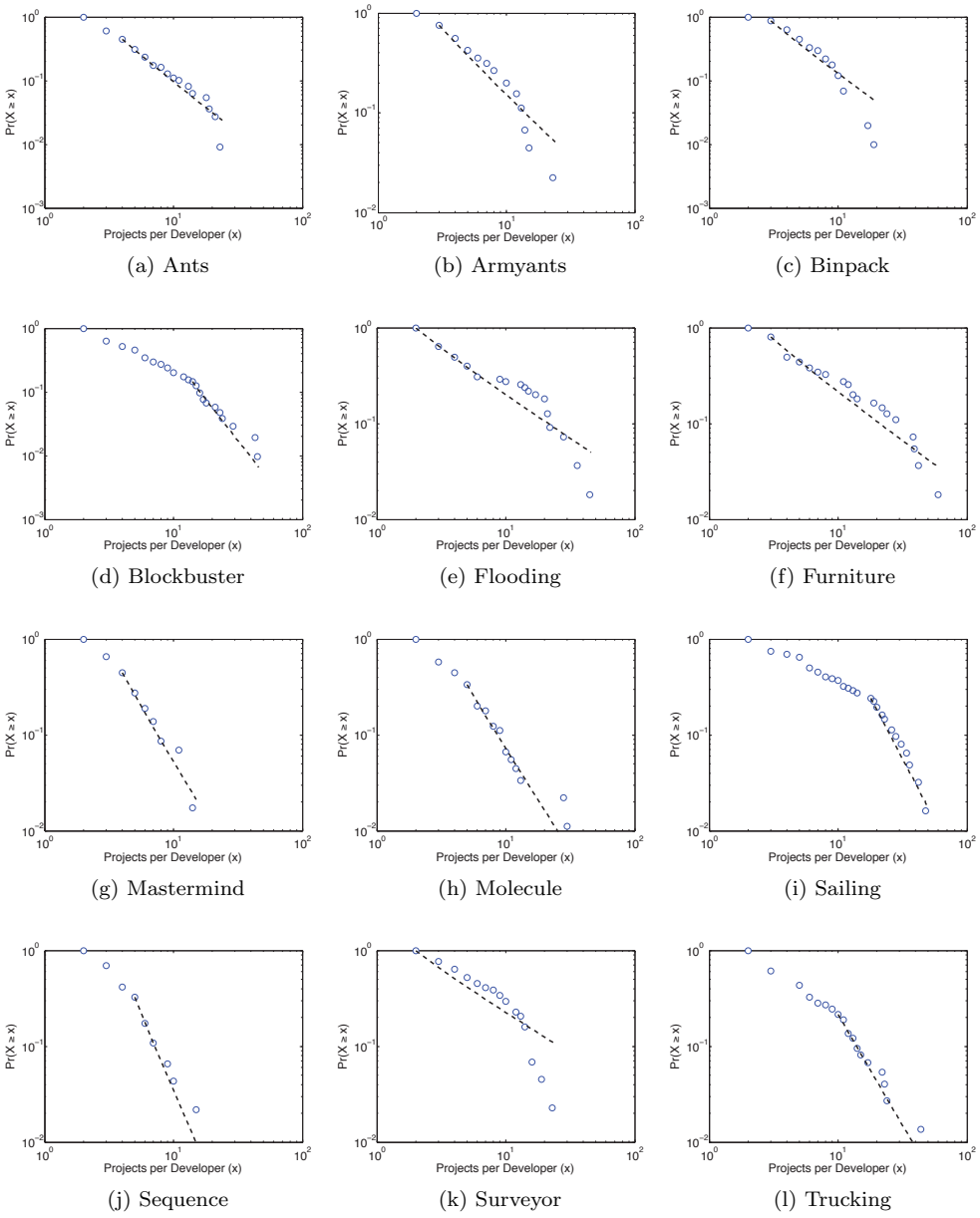
(a) Ants (b) Armyants (c) Binpack

(d) Blockbuster (e) Flooding (f) Furniture

(g) Mastermind (h) Molecule (i) Sailing

(j) Sequence (k) Surveyor (l) Trucking

Figure 1: Developers per project for 12 of the programming contests.



(a) Ants (b) Armyants (c) Binpack

(d) Blockbuster (e) Flooding (f) Furniture

(g) Mastermind (h) Molecule (i) Sailing

(j) Sequence (k) Surveyor (l) Trucking

Figure 2: Projects per developer for 12 of the programming contests.

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	p
Ants	227	1	2.94	0.00
Armyants	71	1	2.07	**0.55**
Binpack	258	1	3.09	**0.16**
Blockbuster	301	1	2.73	0.00
Flooding	229	1	2.63	**0.95**
Furniture	288	1	2.77	0.02
Mastermind	97	1	2.93	**0.12**
Molecule	177	1	2.84	0.08
Sailing	288	1	2.42	0.04
Sequence	66	1	2.87	0.05
Surveyor	165	1	3.06	0.00
Trucking	139	1	2.12	**0.32**

bold values are statistically significant ($p > .10$)

Table 2: Parameters for the developers per project distributions.

Contest Name	n	$\hat{x}_{min}$	$\hat{\alpha}$	p
Ants	110	4	2.55	**0.44**
Armyants	45	3	2.21	**0.24**
Binpack	101	3	2.44	0.00
Blockbuster	104	14	3.54	**0.66**
Flooding	55	2	1.89	0.05
Furniture	55	3	1.99	0.04
Mastermind	58	4	3.16	**0.75**
Molecule	90	5	3.06	**0.58**
Sailing	62	18	3.53	**0.96**
Sequence	46	5	4.02	**0.61**
Surveyor	44	2	1.82	0.01
Trucking	74	10	3.25	**0.55**

bold values are statistically significant ($p > .10$)

Table 3: Parameters for the projects per developer distributions.

a new, better submission is introduced, then it would replace some of the other approaches in use. Therefore, if you look at the submission diversity before and after the introduction of this improvement, then it would be expected that the variety of approaches being used would decrease. From looking at Figure 3, the best-so-far solutions coincide with periods of decreased submission diversity. The submission diversity was calculated by looking at the 10 submissions by unique authors before and after each submission and counting the number of unique approaches. A negative change in submission diversity means that the submission population became more homogeneous and a positive change in diversity means the population became more heterogeneous.

Figure 4 shows the difference in the average change in diversity for the best-

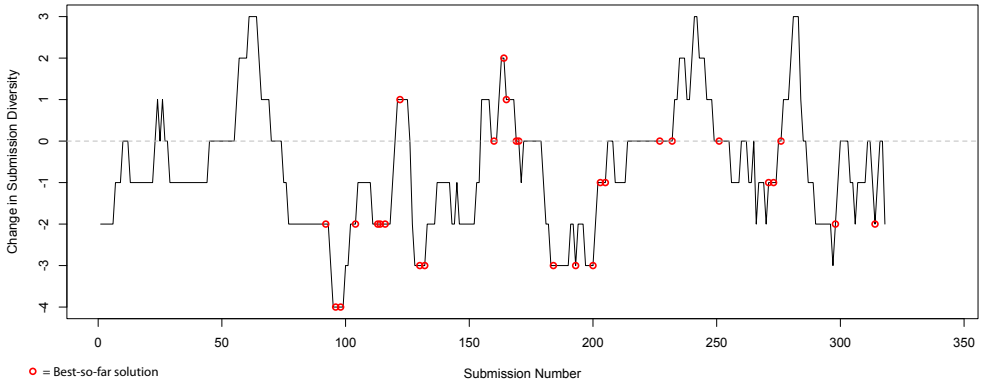


Figure 3: Creative destruction in the Mastermind contest. The red dots show the locations of the best-so-far submissions.

so-far solutions versus all of the other submissions. The majority of the contests show a greater decrease in diversity for the best-so-far submissions. Also, for five of the twelve contests, there is a statistically significant difference (using a t-test) between the change in submission diversity of the best-so-far submissions and the other submissions. It is also important to note that there is a tendency of the submission diversity to decrease over the duration of a contest. As the best approaches are discovered, less effort is placed on exploration of new techniques and more placed on the exploitation of the best solutions.

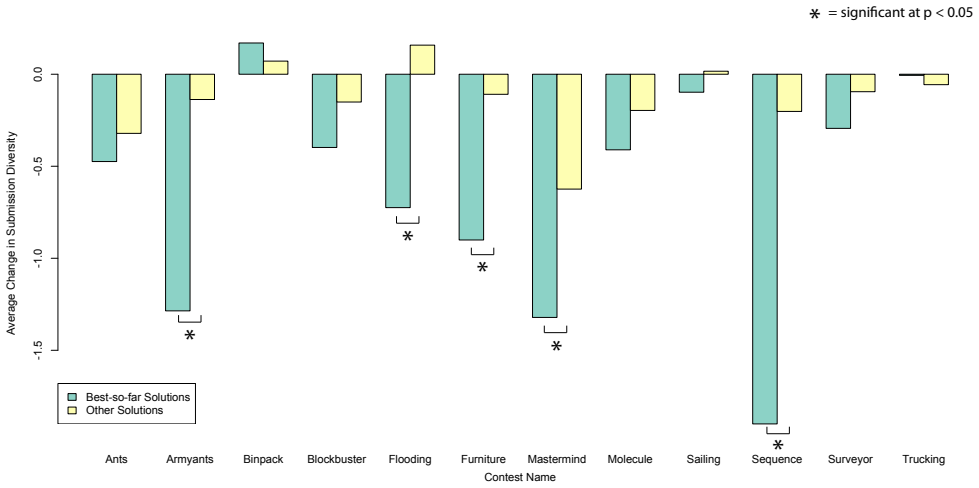


Figure 4: Creative destruction in twelve of the contests.

5 Discussion

This analysis provides a quantitative way of exploring the effect of problem characteristics on the ability to innovate. Problem modularity and size were significant predictors of innovation, although modularity was significant for both dependent variables, while size was significant for only one. The regression models that include problem modularity achieved higher R-squared values, than those achieved using the problem size. While these differences in R-squared values were modest, they provide quantitative evidence that problem modularity is as important as problem size in fostering solution improvements. Naturally, these types of problem characteristics are expected to play a part in the innovation process, but this type of empirical data analysis allows for comparison between difference aspects of the problem-solving task.

The analysis of the distributions of developers per project and projects per developer show that heavy tailed distributions can emerge in a short period of time. It also shows that the MATLAB programming contests have similar patterns as what is seen in the open source movement. Therefore, it may be a more controlled way of studying open source software and its evolution.

Finally, the drop in submission diversity after the introduction of a best-so-far solution provides evidence of creative destruction. The contests may provide insight in how creative destruction occurs in the real world. This data analysis is still work in progress. Hopefully this type of data analysis will give further clues about the innovation process of groups of collaborating problem-solvers.

References

[1] "MATLAB on-line programming contest home page." [Online]. Available: http://www.mathworks.com/contest/

[2] N. Gulley, "Patterns of innovation: A web-based MATLAB programming contest," *Extended Abstracts of CHI 2001*, pp. 337–338, March-April 2001.

[3] J. Frederick P. Brooks, *The Mythical Man-Month.* Boston, MA: Addison-Wesley, 1995.

[4] G. Madey, V. Freeh, and R. Tynan, "Understanding oss as a self-organizing process," *The 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering (ICSE2002)*, 2002.

[5] J. A. Schumpeter, *Capitalism, Socialism and Democracy.* New York: Harper and Row, 1942.

[6] A. Clauset, C. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.