ISCA 20th International Conference on Computers and Their Applications (CATA-2005)

March 16-18, 2005

New Orleans, Louisiana, USA

RELATIONSHIP BETWEEN CODE METRICS AND CHANGE HISTORY

Based on Data from Open-Source Mozilla Project

James Fawcett, Ph.D.
jfawcett@twcny.rr.com
(315) 443-3948

Murat Gungor, MSCS
mkgungor@ecs.syr.edu
(315) 443-4003

Arun Iyer, MSCS
aviyer@ecs.syr.edu
(315) 443-4003

Kanat Bolazar, MSCS
kanat2@yahoo.com
(315) 443-4003

Electrical Engineering and Computer Science
Syracuse University
Syracuse, New York 13244, USA

Abstract

This paper studies relationships between code metrics and change count histories for a large open-source project, Mozilla. We examine several structural and code property metrics and construct statistically significant relationships with code base changes. Our results provide one step toward tool support for software managers to detect problems in large development projects.

Keywords: Change analysis, metrics, open-source

1. INTRODUCTION

This paper studies relationships between code metrics and change count histories for a large project. The analysis is file based. That is, we compute a variety of metrics for each source code file in several large libraries from the open-source Mozilla project, and relate them to the number of cumulative changes for each of those files in several builds. We use files because changes are recorded for files in the data we examined; and because files are the units of configuration management in large software projects.

Others, German [1] and [2], Huntley [3], have examined open-source project data but we've found no modeling of the reliability of metrics to measure potential for change, as reported here. Graves, et. al. [4] analyzed relationships between change metrics and predicted faults, using data from a telephone switching system. Our focus is on modeling change history using code metrics.

We chose Mozilla because it is large[1], accessible, and has provided a wealth of change data in its source code repository (CVS) database. Most of the data presented here is drawn from the Windows build of Mozilla [5], for several releases, spaced approximately one year apart.

We downloaded the CVS archive for these releases and, using make tools provided by the Mozilla project, built Windows executables for one specific release, 1.4.1. During the build, we captured all the files being compiled and used that file set for our analysis of variation of metrics with time[2].

In the next section, we show how changes and the total number of files have grown over the life of the entire[3] Mozilla project. We then show how changes, number of files, estimated defect counts, and metric values, have varied over the four releases and one CVS check-out we analyzed.

In the third section we analyze four libraries and the entire Windows build for the 1.4.1 release, 10 October 2003. The analysis uses Multiple Linear Regression (MLR) [6] [7] to model production of changes as functions of the metrics set, described below. The results are evaluated in terms of resulting t-test and adjusted R-square statistics. In all analyses, we find statistically significant relationships between some of the metrics used and change history, for each of the four libraries, and for the entire Windows build. The results show, however, that not all of the change is related to these metrics and is dominated by two of them, fan-out and total lines of code.

2. PROJECT WIDE MEASURE OF SIZE AND CHANGE

All change and defect data were extracted from the CVS change logs of the Mozilla project. Figure 1 shows the number of files and cumulative changes over the lifetime

[1] There are 6193 source code files in the Windows build for version 1.4.1

[2] Mozilla code management is based on libraries that contain files for all supported platforms. We used the output of building 1.4.1 for Windows to identify the source code for the Windows build, and used those files present in each of the other builds to analyze changes in average metric values with time.

[3] Entire means all files and all changes for all of the platforms supported by Mozilla.

of the entire Mozilla project, as of 10 September, 2004. The latest data consisted of 36,800 files, of which, 14,210 are C/C++ source code[4]. Mozilla CVS captures changes, with and without bug numbers. In the metric analysis we count only changes for source code files with an associated bug number in its change log. The numbers for the entire Mozilla project are shown in Table 1.

Table 1 – Cumulative Change Counts
10 September 2004

Changes	All Files	Source Code Files
All Changes	502,753	305,844
Changes with Bug numbers	255,904	156,903

To quantify defects, we counted the number of unique bug numbers for all the changes against a specified file. The results for defects were far less statistically significant than for change counts. We observed aggregate file check-ins with shared logs which may inflate estimated defect counts. So, we believe our construction of defect counts, based on this data, is not very accurate, but find no other data in the CVS change logs or Bugzilla database used by the Mozilla team, that relates to defects. For that reason, we will not consider defects further in this paper, other than to show variation of our definition over several releases in Figure 5.

For large projects, like Mozilla, the volume of files and their rate of change make it virtually impossible for one person to understand the structure and semantics of the entire project. It is crucial that the tools we develop to analyze systems of this size do not require detailed understanding of all the lines of code in the project, or even the lines of code in a single build for a single platform.

Our goal is to develop tools that program managers and architects can use to understand when a large program is developing problems in its code base. One measure of these problems is the volatility of its changes. Making changes are expensive in schedule time and staffing costs. Managing change is an essential part of managing budget and schedule.

We show, in Figure 1, the history of cumulative change, number of source files in the code base, and, in Figure 2, the number of changes per file, as a function of time over the entire history of the Mozilla project [8], starting on 28 March, 1998, as measured from the first CVS check-in, through our last data extraction on 10 September, 2004.

[4] Files that are not source code include files with extensions ini, mk, idl, html, css etc.

The level of effort required to manage hundreds of thousands of changes, as experienced in the Mozilla code base since the project began six years ago, would be difficult to sustain for any project, but especially for projects that do not follow the open-source model with large numbers of volunteer developers. Project managers need mechanisms to predict and control change. We show, in the following, that changes experienced by the Mozilla code base are significantly related to only a few of the metrics analyzed.

Figure 1 - Total Buggy Change Count, Number of Source Files

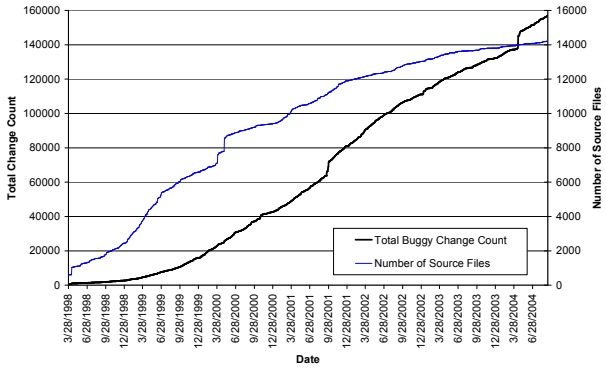
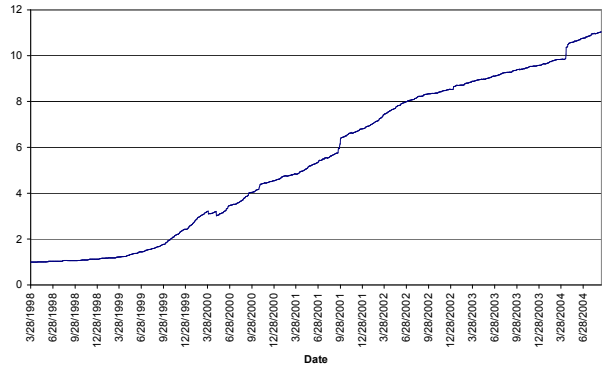


Figure 2 - Average Number of Buggy Changes over Life of All Alive Source Files



3. METRIC ANALYSIS

For analysis of the relationship between various metrics and change counts we use only source files for the Windows platform build. There are 6,193 source code files in release 1.4.1 for Windows, for example, and we count the changes with bug numbers for those files.

Several of the metrics we examine, e.g., Fan-In, Fan-Out, and size of strong components (groups of mutually dependent files), are based on the dependency graph between files in a project [9]. This dependency graph captures static type and function calling dependencies. It is built using a dependency analyzer tool we developed based on a modest subset of the C/C++ language grammar. We also examine size and complexity metrics evaluated with a tool used by one of us for grading

graduate software design project assignments[5]. We have also looked at maximum function cyclomatic complexity, maximum function size per file, average function size per file, and complexity per line of code, but settled on the metrics in Table 2 as being the best measures of those we examined.

The modeling tool used here is Multiple Linear Regression Analysis (MLR), which attempts to predict historical change counts as a linear combination of the metrics in Table 2.

Table 2 – Metrics used in this Analysis

Fan-In: Number of files that depend on a given file.
Fan-Out: Number of files a given file depends on.
Instability[6]: I = Fan-Out / (Fan-Out + Fan-In)
Size of strong component (SCSize): Number of files that have mutual dependencies with a given file. Every file in a strong component has a direct or indirect dependency on every other file in the component.
Global declaration count per file (GblObjDec): The number of global data declarations in a specified file.
Average cyclomatic complexity[7] per file (AvgCC): The number of regions defined by the control flow graph of a function, e.g., one plus the number of loops and branches[8] per function, averaged over all the functions in each file.
Total lines of code (TLOC): The total lines in source file, including white space, declarations, executable code, and comments, e.g., every line in each function body, summed over all the functions in each file.
Lifetime The number of days that the file has been under CVS control.

Clearly change counts are not synonymous with quality. A file with excellent quality may change because its requirements change or because the interface presented by some file on which it depends has changed. Also, a file with low quality may not change often because it is so big and complex that developers are reluctant to make any but the most urgent changes to its code.

However, change effort is directly related to a program's ability to meet its budget and schedule obligations [10]. It would be interesting to examine change effort directly, but the data available in Mozilla CVS does not support deriving effort, only change count, so we have used that information throughout this paper.

3.1 ANALYSIS OF WINDOWS BUILD RELEASES

In this section we analyze five Mozilla builds for the Windows platform, separated by approximately one year, each.

	Release	Date
1	0.6	06 December 2000
2	0.9.7	20 December 2001
3	1.0.2	07 January 2003
4	1.4.1	10 October 2003
5	CVS Check Out	10 September 2004

First, we show the number of files, cumulative changes, and defects, for each build, in Figures 3, 4, and 5.

Figure 3 - Number of Files in Library by Release

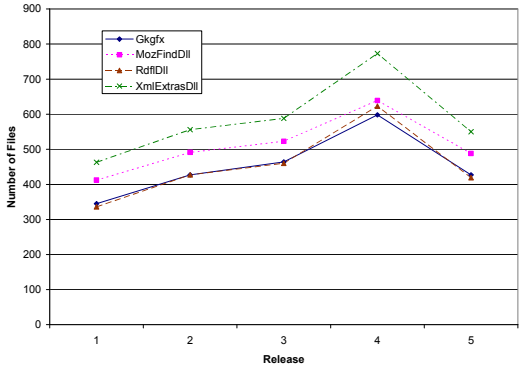
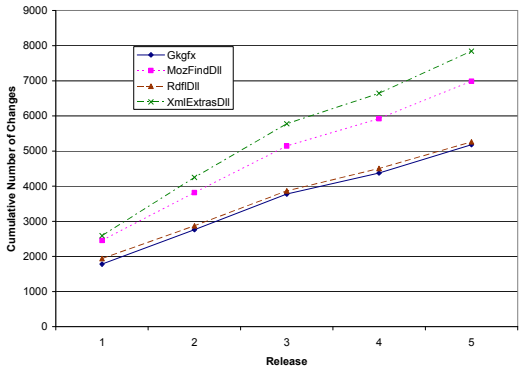


Figure 4 - Cumulative Changes in Library by Release



[5] We think of metrics global object declaration count, average cyclomatic complexity, and average function size per file, as measures of code quality, but have not demonstrated that they are associated with defect counts, so we avoid use of that term in the paper.
[6] Similar to the class-based model of Martin [11].
[7] Our complexity measure is similar, but not identical, to the McCabe Cyclomatic Complexity metric.
[8] This includes continue, break, and goto statements.

Figure 5 - Defect Count by Release

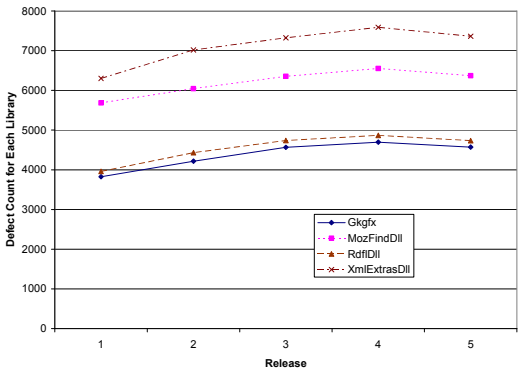
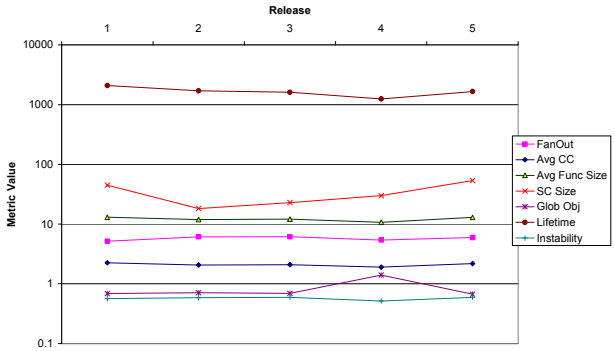


Figure 6 shows that metric values are fairly stable over the four years of code base evolution captured by these releases. It would be interesting to observe a project where these measures were used to direct corrective effort to see if corrections had a significant affect on their average values.

Figure 6 - Variations of Metric Averages over all Files in Gkgfx Library By Release



3.2 MULTIPLE LINEAR REGRESSION

Our goal is to determine if the metrics, shown in Figure 6, are related to changes shown in Figure 4. In Table 3, we show a sample set of results from a Multiple Linear Regression Analysis (MLR) for the MozFindDll library. This models cumulative change for all files in this library, using MLR, as a function of the eight metrics shown. The Adjusted R Square statistic indicates that this model accounts for about 73 percent of the actual changes observed. The t statistic magnitudes greater than 2 indicate that the metrics Fan-out, Average Cyclomatic Complexity, and TLOC are statistically significant. Taking into account typical values for each of these metrics and the coefficients from the model, we find that Fan-out and TLOC dominate predicted change.

In Figure 7, you will see plotted the actual changes, and the changes predicted by the linear regression model. We have sorted the file sequence by actual change value to make the plot easier to interpret. When the model predicts small change the actual changes tend to be small and when predicted changes are large the actual change tends to be large. The results are similar for each of the four libraries examined, as indicated by the plots in Figures 8 through 10. When we analyze the entire Windows build, the Adjusted R-Square statistic and actual versus modeled change improves.

Table 3 – Results of Multiple Linear Regression MozFindDll, Release 1.4.1

SUMMARY OUTPUT

Predicted and Actual Changes for Mozilla's MozFindDll Library
Release 1.4.1, 10 October 2003

Regression Statistics	
Multiple R	0.858457045
R Square	0.736948497
Adjusted R Square	0.731926034
Standard Error	9.752451239
Observations	428

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	8	111644.6583	13955.58229	146.7304964	1.9991E-116
Residual	419	39851.21786	95.11030517		
Total	427	151495.8762			

	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	2.630161	3.649069	0.720776	0.471449
FanIn	-0.001011	0.046063	-0.021958	0.982492
FanOut	0.949522	0.078886	12.036699	0.000000
AvgCC	-0.545876	0.165891	-3.290572	0.001084
SCSize	0.001222	0.003771	0.324012	0.746090
GObjDeclCount	0.023998	0.101890	0.235531	0.813912
TotalLOC	0.016478	0.001033	15.944396	0.000000
LifeOn_2003_10_10	0.000095	0.001898	0.050215	0.959975
Instability	-2.514524	1.726392	-1.456520	0.145998

In the correlation matrix, given in Table 4, we see that Fan-out is most strongly correlated with predicted change, and also to a lesser extent, correlated with TLOC and Average Cyclomatic Complexity. Predicted change most strongly correlates with TLOC, followed closely by Fan-out.

Table 4 Correlation Matrix for MLR Model MozFindDll, Release 1.4.1

Predicted and Actual Changes for Mozilla's MozFindDll Library
Release 1.4.1, 10 October 2003

	FanIn	*FanOut*	*AvgCC*	*SCSize*	*GObjDeclC*	*TotalLOC*	*LifeOn_20(*	*Instability*	*Cumulative*
FanIn	1								
FanOut	0.071353	1							
AvgCC	-0.000963	0.39313	1						
SCSize	0.099422	0.423766	0.306617	1					
GObjDeclCount	0.059384	0.21411	0.025957	0.139319	1				
TotalLOC	0.176208	0.588954	0.593417	0.341359	0.140996	1			
LifeOn_2003_10_10	0.183703	-0.090313	0.067559	0.030067	0.008245	0.147321	1		
Instability	-0.372109	0.413485	0.197215	0.166101	0.025796	0.143086	-0.242959	1	
Cumulative Change	0.153469	0.731609	0.412359	0.357888	0.180698	0.784982	0.053095	0.201202	1

Figure 7 - Predicted and Actual Changes for Mozilla's MozFindDll Library
Release 1.4.1, 10 October 2003

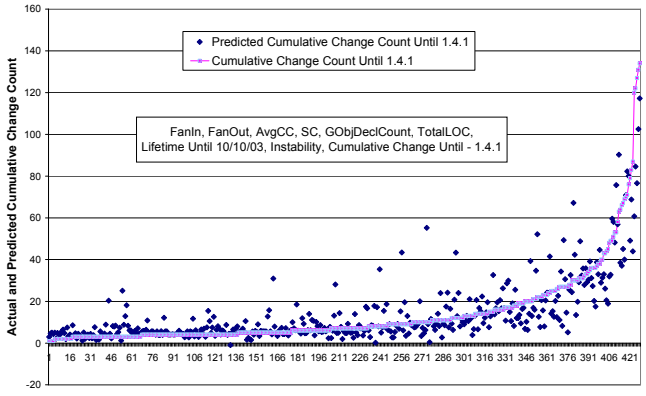


Figure 8 - Predicted and Actual Changes for Mozilla's XmlExtrasDll Library Release 1.4.1, 10 October 2003

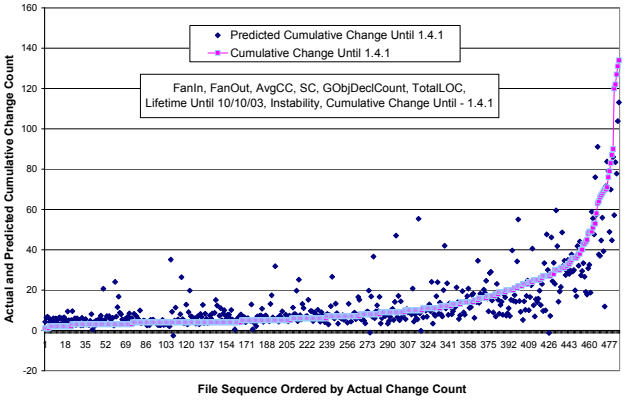


Figure 9 - Predicted and Actual Changes for Mozilla's GKGFX Library Release 1.4.1, 10 October 2003

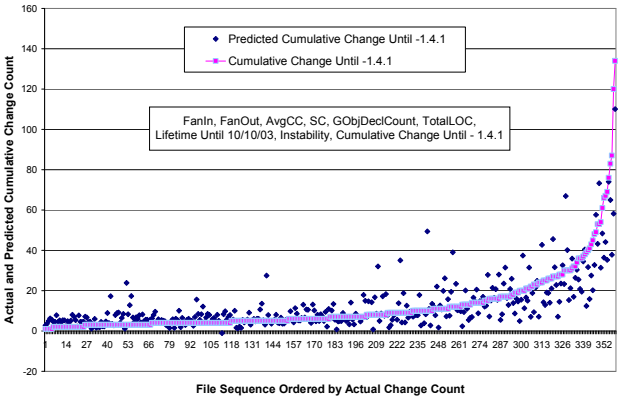
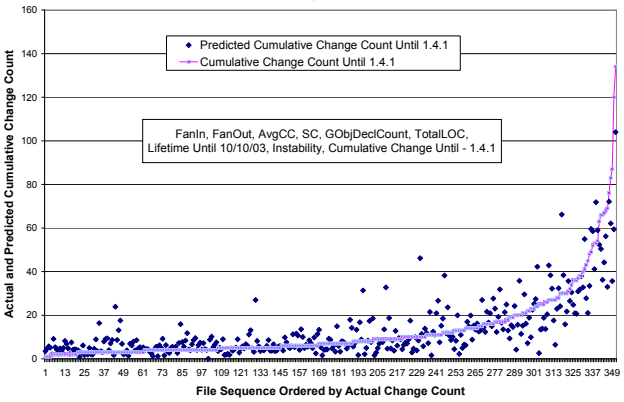


Figure 10 - Predicted and Actual Changes for Mozilla's RDFLDLL Library Release 1.4.1, 10 October 2003



In Table 5 we show results of an MLR analysis on the entire Windows build for Mozilla, release 1.4.1. The model accounts for about 80 percent of the variation in cumulative change count and Fan-out, Average Cyclomatic Complexity, number of Global Object Declarations, Total Lines Of Code, and Instability are all statistically significant.

Table 6 shows correlation matrix resulting from this MLR analysis. It is interesting that Fan-out correlates more strongly with change than total size. Figure 11 illustrates predicted and actual changes for this MLR analysis.

Figure 11 - Predicted and Actual Changes for Windows Build of Mozilla Release 1.4.1, 10 October 2003

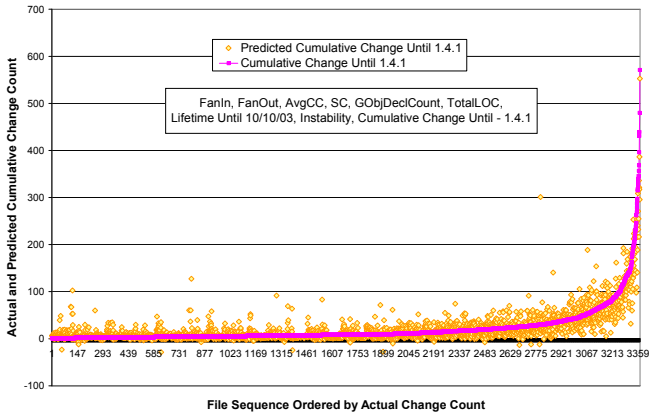


Table 5 – Results of Multiple Linear Regression
Windows Build of Mozilla, Release 1.4.1

SUMMARY OUTPUT — Predicted and Actual Changes for Windows Build of Mozilla Release 1.4.1, 10 October 2003

Regression Statistics	
Multiple R	0.901019564
R Square	0.811836255
Adjusted R Square	0.811388379
Standard Error	17.03564781
Observations	3370

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	8	4208412.589	526051.5737	1812.637741	0
Residual	3361	975406.8887	290.2132962		
Total	3369	5183819.478			

	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	-0.890092	2.180555	-0.408195	0.683156
FanIn	0.000173	0.007200	0.024007	0.980849
FanOut	1.371579	0.026634	51.496815	0.000000
AvgCC	-0.873727	0.090125	-9.694663	0.000000
SCSize	-0.002014	0.000218	-9.240209	0.000000
GObjDeclCount	-0.264539	0.034726	-7.617985	0.000000
TotalLOC	0.018727	0.000542	34.543551	0.000000
LifeOn_2003_10_10	0.003227	0.001258	2.565292	0.010352
Instability	-5.578067	0.946077	-5.895994	0.000000

Table 6 – Correlation Matrix for MLR Model
Windows Build of Mozilla, Release 1.4.1

Predicted and Actual Changes for Windows Build of Mozilla Release 1.4.1, 10 October 2003

	FanIn	*FanOut*	*AvgCC*	*SCSize*	*GObjDeclC*	*TotalLOC*	*LifeOn_20(*	*Instability*	*Cumulative*
FanIn	1								
FanOut	0.036412	1							
AvgCC	-0.00219	0.260892	1						
SCSize	0.056421	0.34662	0.212668	1					
GObjDeclCount	0.016996	0.12169	0.088853	0.077633	1				
TotalLOC	0.08118	0.721062	0.406104	0.249548	0.188281	1			
LifeOn_2003_10_10	0.125683	0.035249	0.134067	0.11667	0.046017	0.152023	1		
Instability	-0.198229	0.402716	0.22057	0.184166	0.020203	0.184365	-0.157599	1	
Cumulative Change	0.064236	0.850391	0.218924	0.216204	0.082584	0.795019	0.090537	0.240712	1

Note that Cumulative change correlates most strongly with Fan-out, then GblObjDec and TLOC.

In Table 7, we show the significant metrics for each library and the entire Windows build, along with their Adjusted R-Square statistic for the fit to each library. Note that the significant metrics were not the same for each library. Only Fan-Out and TLOC are significant for all analyses.

Table 7 – Summary of MLR Statistics

Comparison of Multiple Linear Regression Analysis Results									
Library	Adj R-Sq	Fan-In	Fan-Out	Avg CC	SC Size	GblObjDec	TLOC	Lifetime	Instability
Gkgfx	0.65353		significant	significant			significant		significant
MozFindDll	0.7325		significant	significant			significant		
RdfIDll	0.69269		significant				significant		significant
XmlExtrasI	0.70157		significant	significant		significant	significant		significant
All Mozilla	0.80665		significant	significant		significant	significant		significant

Note: Blank entries indicate that metric had no significant affect on predicted change

4. DENSITY METRICS

The results above seem to indicate that, while Fan-out and TLOC dominate the predicted changes, Average Cyclomatic Complexity, Global Object Declarations, and Instability also contribute significantly to the modeled results.

Because of the correlations of Fan-out with TLOC, we decided to look further by attempting to normalize out the effects of size as measured by TLOC. To do this we used the metrics Fan-out/TLOC, AvgCC, GblObjDec/TLOC, Lifetime, and Instability to predict Cumulative Change/TLOC. The results were illuminating. We found that the density model [12] does a very poor job of predicting cumulative change.

The resulting Adjusted R Square statistic was 0.1567, indicating that the density metric set is a very poor predictor of cumulative change density. We interpret this to mean that Fan-out and TLOC metrics are the only effective predictors of cumulative change, and because they are relatively correlated, the density Fan-out/TLOC is not a very strong predictor either.

To see which is stronger, we built MLR models for the Windows Build of Mozilla, using Fan-out alone and TLOC alone and found that Fan-out describes the change data better with an Adjusted R Square statistic of 0.72308, while TLOC had an Adjusted R Square of 0.63194. So, Fan-out is the stronger predictor of the two.

5. CONTRIBUTIONS

There has been other recent work that focuses on dependency analysis, investigation of open-source projects, and analysis of change histories. Godfrey and Lee [8] examine calling relationships between subsystems of Mozilla and VIM text editor. Tzerpos, in an early paper [9] develops source code file dependency structure based on include relationships. Huntley [3] builds models of the benefits of learning in an open-source environment. The work of German et. al. [1] and [2] examines changes using non-code based measures. Graves, et. al. [4] study the fault potential of modules (groups of files) for a telephone switching system, based on the number of their changes. De Lucia et. al. [10] study the relationship between effort and non-dependency code metrics for work packages undergoing Y2K conversions. Gill and Kemerer [12] investigate the relationship between cyclomatic complexity and maintenance productivity in applied hours for a small system.

The contribution of this paper is to relate propensity for changes to files based on metrics derived from both their dependency graph – Fan In, Fan Out, Strong Component sizes, and a file-based instability metric – and from attributes of the file's code – size, average cyclomatic complexity, and global declaration counts. We developed tools that evaluate dependency structure that are robust and efficient enough to successfully analyze thousands of files. All of our results were derived based on code from the open-source Mozilla project. The methods of this paper use Multiple Linear Regression analysis to model the predictive power of several metrics that depend of either dependency structure between files or local properties of the code. We have discovered that Fan-Out is the strongest single predictor of change for the Mozilla project, even stronger than file size. This is, we believe, a new result.

6. CONCLUSIONS

Only Fan-out and Total Lines Of Code (TLOC) are strong predictors of cumulative change for the Mozilla Windows 1.4.1 build code base. Surprisingly, Average Cyclomatic Complexity (AvgCC), the number of Global Object Declarations (GblObjDec), and size of Strong Components (SCSize) have virtually no modeling power for cumulative change in that code base.

The Mozilla data provides no measure of effort expended to make changes. It would be very interesting to examine a code base for which such data was available. It is possible that complexity, use of global data, and large mutual couplings may be more highly correlated with effort than we found for change.

7. REFERENCES

[1] Daniel M. German, Abram Hindle and Norman Jordan, "Visualizing the evolution of software using softChange," Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 336-341, 2004.

[2] Daniel German, Audris Mockus, "Automating the Measurement of Open source Projects," ICSE 2003, 3rd Workshop on Open Source Software Engineering.

[3] Christopher L. Huntley, "Organizational Learning in Open-Source Software Projects: An Analysis of Debugging Data," IEEE Trans. Engineering Management, vol. 50, no. 4, pp. 485-493, 2003.

[4] Todd L. Graves, Alan F. Karr, J. S. Marron, Harvey Siy, "Predicting Fault Incidence Using Software Change History," IEEE Trans on SE, vol. 26, no. 7, pp 653-661, July 2000.

[5] Mozilla on Microsoft Windows 32-bit Platforms, www.mozilla.org/build/win32.html

[6] Larry Stephens, Advanced Statistics Demystified, McGraw Hill Inc., May 2004.

[7] Schuyler W. Huck, Reading Statistics and Research, Addison Wesley Longman, 2000.

[8] Michael W. Godfrey, Eric H. S. Lee, "Secrets from the Monster: Extracting Mozilla's Software Architecture," Proc. of the Second Intl. Symposium on Constructing Software Engineering Tools (CoSET-00), Limerick, Ireland, June 2000.

[9] Vassilios Tzerpos, "Automatic Source - File Dependency Structure Extraction for C Programs," Proc. of the 1994 Conf. of the Centre for Advanced Studies on Collaborative research, pp. 68-75, 1994.

[10] Andrea De Lucia, Massimiliano Di Penta, Silvio Stefanucci, Gabriele Venturi, "Early Effort Estimation of Massive Maintenance Processes," IEEE Proc. of the Int. Conf. on Software Maintenance, pp. 234-237, 2002.

[11] Robert Martin, Agile Software Development, Prentice Hall, 2003.

[12] Geoffrey K. Gill and Chris F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," IEEE Trans on SE, vol. 17, no. 12, pp. 1284-1288, Dec 1991.