The Impact of Developer Team Sizes on the Structural Attributes of Software

Ahmmad Youssef
Brunel University
London, United Kingdom
Ahmmad.Youssef@brunel.ac.uk

Andrea Capiluppi
Brunel University
London, United Kingdom
Andrea.Capiluppi@brunel.ac.uk

ABSTRACT
It is established that the internal quality of software is a key determinant of the total cost of ownership of that software. The objective of this research is to determine the impact that the development team's size has on the internal structural attributes of a codebase and, in doing so, we consider the impact that the team's size may have on the internal quality of the software that they produce.

In this paper we leverage the wealth of data available in the open-source domain by mining detailed data from 1000 projects in GoogleCode and, coupled with one of the most established of object-oriented metric suites, we isolate and identify the effect that the development team size has on internal structural attributes of the software produced.

We will find that some measures of functional decomposition are enhanced when we compare projects authored by fewer developers against those authored by a larger number of developers while measures of cohesion and complexity are degraded.

Categories and Subject Descriptors
D.1.5 [Programming Techniques]: Object-oriented Programming

General Terms
Measurement, Design, Economics, Human Factors.

Keywords
Open Source Software Development Process, Complexity Metrics

1. INTRODUCTION
To varying extents, the real and perceived success of an organisation depends on the quality - functional and non-functional - of the software it produces, commissions or purchases. There are many examples of organisations suffering significant loss resulting from poor software processes negatively affecting the quality of the software, eventually impacting an organisation's stakeholders through failed projects or serious software defects. The impact may lead to significant financial or reputational loss and can even be serious enough to cause an organisation to fail. At the very minimum, poor quality can prove a costly drain on resources. In his book 'The Economics of Software Quality' Capers Jones found that one half of most software development project budgets and two thirds of the typical development team's time are spent fixing poor quality [3]. Poor quality also leads to increased cost-to-change which can handicap an organisation's competitive position with a reduced ability to react to an increasingly dynamic world.

Clearly there is a need for a continual drive to understand the factors that can have a material effect on software quality. Software metrics will continue to play an important role in this process as they embody an empirical approach to software engineering that, if used appropriately, can lead to a significant reduction in the implementation and maintenance costs of the final software product. There are three general classifications of software metrics – product metrics, process metrics, and project metrics [4]. Our interest lies in product metrics and, more specifically, internal structural metrics.

One key factor that is clearly within the sphere of influence of management is the size of the development teams. While there has been significant work measuring the impact of team sizes on software process metrics (namely productivity) and limited work measuring impact of team sizes on external software metrics (namely fault-proneness), there has been no research investigating the impact of team sizes on the structural attributes of a codebase. As we will discuss further, filling this gap in the research would enable us to leverage the vast body of research linking object-oriented structural metrics to some key characteristics that matter greatly to practitioners such as testability and maintainability. In doing so, we will pave the way to gaining a significantly greater understanding of the true impact of team sizes on internal software quality.

2. RELATED WORK
2.1 Team Sizes and External Software Metrics
There has been plenty of valuable in-depth research investigating the relationship between development team productivity and its size. In his popular book 'The Mythical Man Month', Brooks argues that, since software development is a complex task, the communication effort is great and adding more developers can lengthen rather than shorten the time taken to complete a task as it adds an exponentially greater number of necessary communication paths between developers [5].

Roger et al., using data from 130 projects, empirically tested the impact of a number of factors on software development productivity concluding that only team size significantly impacts software development time and productivity. This was since

IWPSE'15, August 30, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3816-5/15/08...$15.00
http://dx.doi.org/10.1145/2804360.2804365

independently confirmed using other empirical methods (other papers) [6].

Although the emphasis in the research community has largely been on establishing the link between team sizes and productivity, there has been some work linking team sizes to measures used as a surrogate for software quality. Nagappan leveraged data from Microsoft's Windows Vista project to establish that metrics based on organisational structures – of which team sizes were one aspect - are a significant predictor of software failure-proneness. [7]

2.2 Internal Software Metrics

The study and application of internal software metrics dates back to the mid-1960's when the primitive Lines of Code metric was routinely used as the basis for measuring software development productivity (LoC per month) and quality (defects per KLoC).

In 1971 Akiyama proposed the use of metrics for software quality prediction proposing a regression-based model for module defect density (number of defects per line of code) where line of code was used as a crude indicator of complexity [19]. This was one of the earliest attempts, albeit a simplistic one, to extract an objective measure of software quality through the analysis of observables of the system. With the increasing diversity of programming languages, it became necessary to introduce a more nuanced model of software complexity.

McCabe and Halstead made significant contributions but with the increasing adoption of Object-Oriented (OO) programming languages, Chidamber and Kemerer argued that this new development necessitated measures that could guide organizations to its successful adoption. This fact, coupled with criticisms of existing metrics suites, saw the development of the Chidamber and Kemerer (CK) metrics suite [19].

Academic efforts to extend, validate and refine complexity metrics have been a dominant feature of metrics research ever since. Basili et al., motivated by the desire to leverage software metrics to provide guidance to the areas of a system where testing efforts are best spent, built on Henry's research [18] to establish the utility of the Chidamber and Kemerer software metrics suite as a predicator of fault prone software classes. This was achieved through the diligent assembly of eight software development teams and a thorough regression analysis to establish relationships between OO metrics and observed defects [9].

These are, by no means, the only studies of this nature. Subramanyam et al. conducted similar work with access to a large number of in-house developed codebases and were able to control for programming language and software size, confirming the results obtained by Basili et al. – results which were further validated in a multitude of similar studies, each adding its own unique dimension, whether on the analysis side or the case study subject [10][11][12][13][14].

More recently Saberwal et al. employed similar regression models to correlate CK metrics with bad code smells driven by the desire to guide refactoring efforts to where they are most needed [15]. Badri et al., using similar techniques, concluded that a correlation exists between LCOM and unit test coverage, validating the use of OO metrics as a predictor of the testability of classes [16].

3. RESEARCH PROBLEM

The objective of this research is to establish the impact that a development team's size has on the structural attributes of a codebase. The main caveat when embarking on such a study is that we must remain conscious that codebases with a larger team size are more likely to exhibit higher complexity than codebases with fewer developers simply due to the fact that the larger team is likely to have gone through more iterations of development as more complex functionality is implemented. To elaborate, when using version control systems, it is usual to build functionality iteratively through repeated modification of source files. It has been proven that, as software projects evolve, iterations of a codebase tend to exhibit growing complexity [8]. A key part of our approach to solving our research problem is to isolate and remove any impact that this effect may have on the metrics of an evolving codebase and to observe the impact of the development team size alone. We detail our approach to this particular challenge in section 4.4.

4. METHODOLOGY

4.1 Data Set

Given the wealth of project data accessible in the open source space, this was decided to be the best source of raw data for analysis. There are a number of popular open source project hosts (or 'forges') – GITHub, SourceForge, and GoogleCode. In terms of developer activity, GITHub is the most popular, followed by SourceForge and then GoogleCode [1]. Each of these forges has a unique and varied make-up of languages constituting its project population. It was considered that Java projects would be the most preferable to study given the myriad of available static code analysis tools readily available as well as the large number of projects available for study. Furthermore, Java consistently rates as the Object Oriented language with the highest adoption rates [2], an important consideration when bearing in the mind the need to ensure the relevance of this research to practitioners.

Table 1 A summary of the CK metrics and guidelines

Metric	Full Name	Attributes Measured
CBO	Coupling Between Objects	Count of other objects to which the object being considered is coupled. A high number can indicate poor encapsulation, a low level of reusability, and create difficulties in modification or testing.
DIT	Depth of Inheritance tree	A measure of complexity as measured by the number of parent classes from which a class may inherit behavior. A high number can point towards excessive design complexity.
LCOM	Lack of Cohesion of Methods	Measurement of the disparateness of functionality within an object. A high number can point towards poorly designed objects that do not adhere to the "single responsibility principle".
NOC	Number Of Children	A measure of reuse and abstraction. A high number can point towards poor design and diluted abstraction.
RFC	Response For a Class	Count of methods which may be executed in response to a message. High numbers may highlight objects with undue complexity complicating build, maintenance, and testing.
WMC	Weighted Methods per Class	An indicator of the complexity of a class through the method count in that object. A high number can indicate undue complexity and limited scope for re-use.

GoogleCode was the forge selected for study both for its popularity and high level of Java adoption rates. One final benefit of specifically mining GoogleCode was that project administrators can choose from among three available version control systems – Subversion, GIT, and Mercurial. This ensured that our toolchain needed to be compliant with each version control system meaning that, as GoogleCode itself prepares for eventual shutdown, the toolchain is equally suited to be re-used to mine GitHub (which uses GIT) or SourceForge (which uses Subversion and Mercurial).

4.2 Metrics Suite

The popular metrics suite proposed by Chidamber and Kemerer [12] is both well understood and has a significant supporting body of research. The details of this metric suite are outlined in table 1.

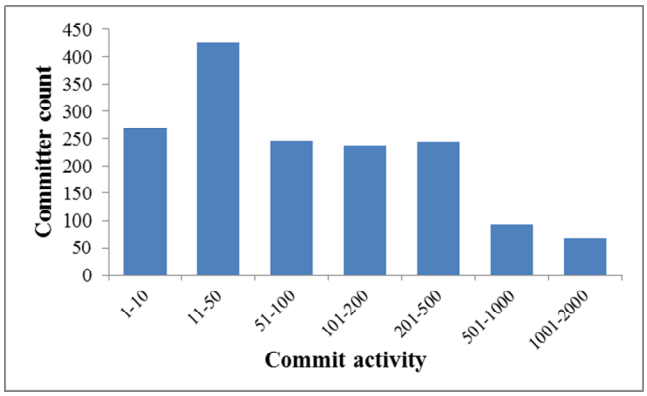
4.3 Defining the Development Team

There are a number of possible definitions of a software development team. Both Capra et al. and Smith et al. consider a team to consist of all developers to have worked on a codebase for any length of time [24][25] while Nagappan et al. (using data from IBSG [26]) consider the development team to also include management, administration and operations personnel.

Given the context of mining open source repositories, we favour defining the team size as the cumulative total of all unique committers present in the revision history in the version control system of a given project. Our view is that this definition is consistent with the prior art, simple to measure and reproduce, and elegantly allows us to capture the number of unique development design approaches that may have influenced the evolution of a codebase.

There are some potential limitations to this approach, most notably that we do not distinguish between frequent committers and causal (infrequent) committers. Figure 1 shows the relative activity levels of the committers in our data sample and, while it is true that the majority of commit activity takes place by a minority of committers, clearly the majority of committers do make a significant contribution and cannot be discounted.

Figure 1 The number of committers exhibiting a activity in a defined range against the number of commits in that range.



4.4 Data Analysis Techniques

In order to investigate the relationship between team sizes and internal quality metrics through mining open source repositories, there is a large amount of data that needs to be collated and analysed. This data essentially takes the form of a large population of file-level CK metrics along with meta-data associated with each file revision. This meta-data allows us to establish the number of unique committers to an individual project which is, to all intents and purposes, the project development team size.

Given the above data, we can group metrics together by project team size - irrespective of the individual project from which they came - and consider them distinct populations. For example, if project X and project Y each had *n* unique committers, all metrics belonging to each file within both projects would reside in a single bucket. Using this bucketing approach, we would have a limited number of distinct populations of metrics which could then be compared using statistical techniques.

As mentioned earlier, we must remain conscious that codebases written by larger teams are more likely to exhibit higher complexity than codebases with less developers given that a larger team will solve more complex problems and a codebase will typically go through more iterations (or 'revisions') in the process. This effect is illustrated in figure 2 where averaging metric values across files modified by a given number of committers yields a steady upward on measures of complexity.

Figure 2 Average metric value for files plotted against the cumulative number of committers to have edited the files.

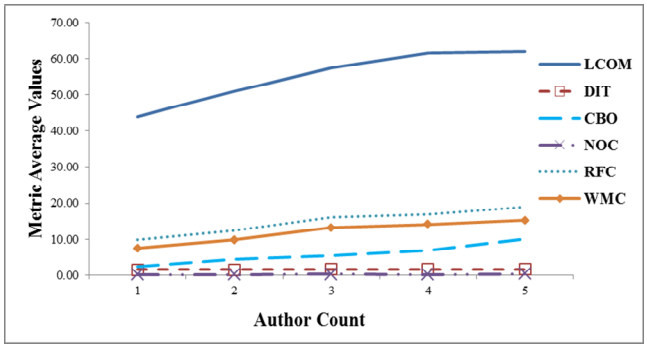
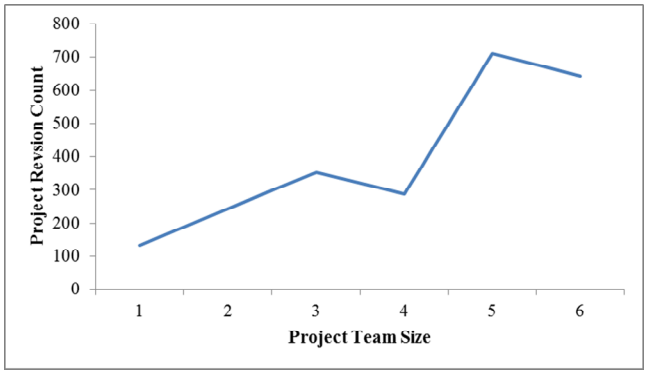


Figure 3 shows the increase in total project revision count against project team size. We can see this impact of this in Figure 4 where we average the metrics values and plot their progression by revision count.

Figure 3 Project revision count plotted against project team size



For this reason, our data analysis will take on an additional dimension alongside team size – namely that of the file revision count. From the meta-data associated with each revision, we can determine how many revisions any one file has undergone. This data will feed into our bucketing process where we can ensure that the population of metrics within a particular bucket only contains those metrics belonging to projects with a particular team size and only from files that have been modified a particular number of times. This approach, illustrated in figure 5, will give us confidence that when comparing our bucketed metric populations, any statistically significant differences are solely down to team size rather than fact that larger teams typically work on larger projects.

When comparing metrics populations bucketed by developer count, the Mann-Whitney test is ideally suited as all metrics populations are independent and consist of continuous data that we found not to be normally distributed.

Figure 4 Average metric values against revision count

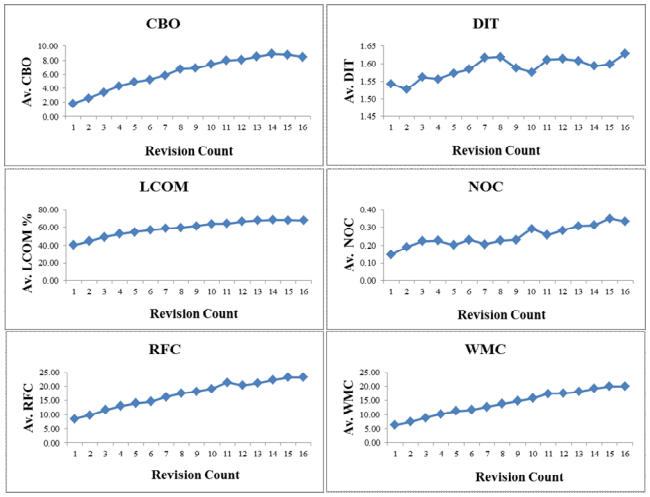
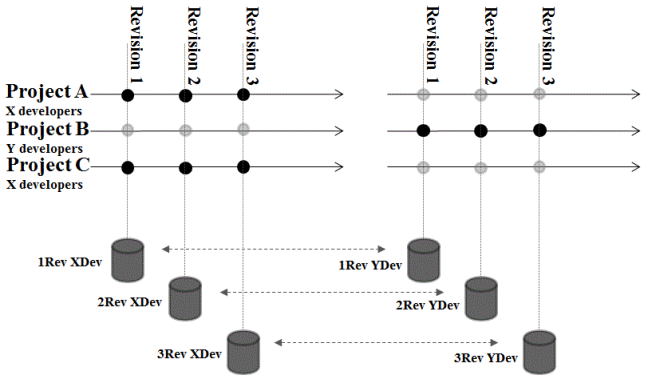


Figure 5 Bucketing approach illustrated



5. DATA COLLECTION

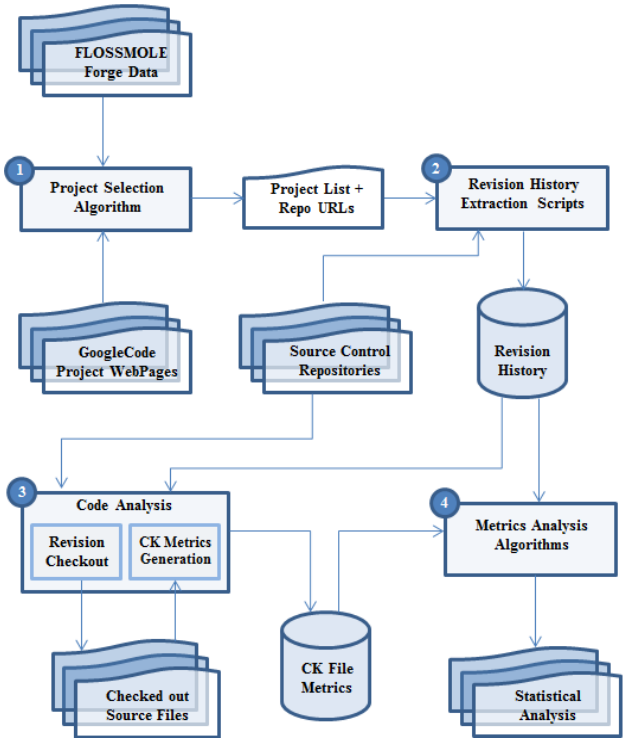
By leveraging the FLOSSmole project [20] we were able to obtain raw data describing, at a project level, details about all projects hosted in GoogleCode. This, along with the GoogleCode project webpages and the associated repositories formed the input into a bespoke toolchain illustrated in figure 6.

It is important to note that the majority of the complexity in this toolchain resides in the Project Selection Algorithm (1), the CK metrics Generation (3), and the Metrics Analysis Algorithms (4). Although the scripts to extract revision history (2) and checkout file revisions (3) may be considered a duplication of effort in prior research (most this work has notably been implemented in CVSAnalY [21]), it was felt that integration challenges in using the such a software package would outweigh the effort we would expend in developing our own bespoke scripts for the relatively simpler parts of the toolchain. For the more complex functionality of metrics generation, an off-the-shelf tool for metrics generation was employed. Our toolchain is illustrated in figure 6.

5.1 Project Selection Algorithm

This component (marked as component 1 in figure 6) is written in Java and takes in flat files made available by FLOSSmole detailing all the available projects hosted by GoogleCode and the 'tags' associated with each project. The project data is used to extract all projects with the tag 'Java' as of May 2012 – yielding us a list of 22594 GoogleCode hosted Java projects. That list was then reduced from 22594 projects to a more manageable subset by employing the pseudorandom 'Math.random' function in Java to select a number between 0-1 to be multiplied by the total number of projects until 1000 projects had been selected. In the process we discard from consideration any projects with no revision history as they represent projects which were not started and have no significance in this study and should not constitute part of our 1000 project sample.

Figure 6 Toolchain to extract and analyse revision based metrics



Once the projects were selected, the algorithm then extracts (via 'screen-scraping') the repository URL from the relevant project's

page on the GoogleCode website. The project list with the associated URLs are consolidated in a single file which is used to drive the next part of the toolchain.

5.2 Revision History Extraction Scripts

The revision history extraction scripts (marked as component 2 in figure 6) are a relatively simple collection of shell scripts (that are runnable on a unix platform) to query, for all the projects in the input file, the relevant version control repositories to obtain the full revision history, storing it in a simple format to allow it to drive the code analysis stage.

5.3 Code Analysis Component

The code analysis component (marked at component 3 in figure 6) comprises, again, fairly simple shell scripts responsible for checking out each version of the project, handing over the heavy lifting of project metrics generation to run a metrics generation tool called 'Understand' by Scientific Toolworks Inc. [28]. The report created by Understand is of a particular format which is then passed through a file parser (written in Java) which extracts the information that is pertinent to our research and stores it in a format appropriate to our metrics analysis component. Understand version 2.6.610 was chosen as it is available on academic license and offers a unix-based command line tool that generates metric reports in an easily parsable format. The calculations used by Understand to generate metric values are in table 2.

Table 2 A description of how CK Metric values are calculated for Java classes by Understand

Metric	Full Name	Calculation for Java classes in 'Understand'
CBO	Coupling Between Objects	Number of other Classes invoked from this class. Library classes not included.
DIT	Depth of Inheritance tree	Number of parent classes in total
LCOM	Lack of Cohesion of Methods	For each member variable calculate the percentage of methods which do not access that variable. Average the percentages to determine LCOM.
NOC	Number Of Children	Count of other classes that directly extend it.
RFC	Response For a Class	Number of total methods including all methods in parent classes (regardless of invocation or visibility).
WMC	Weighted Methods per Class	Count of all methods in that class only (regardless of invocation, visibility, and instance or static).

5.4 Metrics Analysis Algorithms

The metrics analysis algorithms (marked as component 4 in figure 6) are one of the more complex components in our toolchain. It is a software package written in Java, and responsible for retrieving the generated metrics data, implementing the bucketing strategy, and running statistical tests to produce our analysis.

6. RESULTS

6.1 Bucket Populations

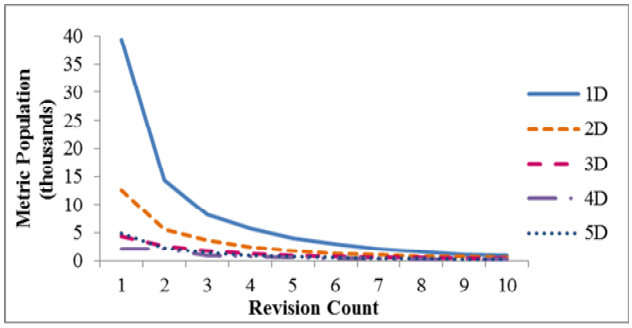
When we conduct a simple Mann-Whitney test for each CK metric type comparing two groups – the first being all metric results belonging to single developer projects and the second group being metrics results from all other projects - we find that the two groups are independent with p-values < 0.05. However, as discussed in detail in section 4.3, when analysing metrics results we bucket our data by revision count and project team size in order to isolate and observe the effect of team size alone.

It is logical that there would be a greater number of metric results pertaining to the lower revision counts as, by necessity, for a file to be revised, say, 5 times, it would have 4 prior revisions. However, it is perfectly normal for a file to only have fewer than 5 total revisions. This is an important consideration as buckets belonging to higher revisions and team sizes will have diminishing populations. Figure 7 and table 3 clearly show this effect. For our analysis, we only consider buckets belonging to team up to 5 developers strong with a maximum of 6 revisions as we see marked drop-off in bucket population sizes as team size and revision counts increase.

Table 3 Bucket population sizes

	PROJECT TEAM SIZE				
Rev	**1 Dev**	**2 Dev**	**3 Dev**	**4 Dev**	**5 Dev**
1R	39460	12524	4372	2060	4962
2R	14236	5552	2630	2169	2316
3R	8291	3662	1758	981	1509
4R	5743	2395	1282	695	973
5R	4024	1705	995	515	687
6R	2985	1311	786	393	528

Figure 7 Average metric values against revision count



6.2 Comparing Lone Developer Projects against Multi-developer Projects

Table 4 details the results from the statistical tests run across each team-size comparison. The first part of the table summarises the comparison results for metrics from single

developer projects against projects authored by two developers. Taking the WMC column as an example, we can see that the 2R bucket (this refers to the bucket of WMC metric values belonging to the second revision of files only) has statistically significantly lower values than in the single developer bucket. To be clear, this means that a Mann-Whitney test yields a p-value of <0.05 and we can observe higher median values in the single developer bucket. This means that we can say with a confidence level of 95% ±5% that files with two revisions and a single developer have higher WMC values than files with two revisions and two developers.
The next bucket down - 1D3R v 2D3R - exhibits no significant difference between the metrics populations for WMC.

In table 4 we can clearly observe that a large number of buckets that show statistical significance across CBO, DIT, LCOM and, to a lesser extent RFC and WMC. In the case of DIT, we can see a progression where more buckets show significance as the team size grows. In the case of other metrics we can see an inconsistent pattern with more buckets showing differences in the 1D v 3D and 1D v 4D tests. Another key observation is that CBO, DIT, RFC and WMC generally trend downwards as project team sizes increase while LCOM increases.

Table 4 Results of Mann-Whitney tests comparing the lone developer bucket of the various revision counts against the corresponding buckets for 2, 3, 4 and 5 developer buckets. The percentages relate to the proportion of the six buckets (1R-6R) that show p-values<0.05 for a particular team size comparison.

1D v 2D	CBO	DIT	LCOM	NOC	RFC	WMC
	17%	0%	0%	0%	0%	17%
	1R -	1R -	1R -	1R -	1R -	1R -
	2R -	2R -	2R -	2R -	2R -	2R >
	3R >	3R -	3R -	3R -	3R -	3R -
	4R -	4R -	4R -	4R -	4R -	4R -
	5R -	5R -	5R -	5R -	5R -	5R -
	6R -	6R -	6R -	6R -	6R -	6R -

1D v 3D	CBO	DIT	LCOM	NOC	RFC	WMC
	83%	17%	67%	0%	33%	33%
	1R -	1R -	1R >	1R -	1R >	1R >
	2R >	2R <	2R -	2R -	2R >	2R >
	3R >	3R -	3R <	3R -	3R -	3R -
	4R >	4R -	4R <	4R -	4R -	4R -
	5R >	5R -	5R -	5R -	5R -	5R -
	6R >	6R -	6R <	6R -	6R -	6R -

1D v 4D	CBO	DIT	LCOM	NOC	RFC	WMC
	83%	67%	67%	0%	33%	17%
	1R -	1R >	1R -	1R -	1R >	1R -
	2R <	2R <	2R -	2R -	2R <	2R >
	3R >	3R -	3R <	3R -	3R -	3R -
	4R >	4R -	4R <	4R -	4R -	4R -
	5R >	5R <	5R <	5R -	5R -	5R -
	6R >	6R <	6R <	6R -	6R -	6R -

1D v 5D	CBO	DIT	LCOM	NOC	RFC	WMC
	17%	83%	0%	0%	33%	50%
	1R <	1R -	1R -	1R -	1R -	1R -
	2R -	2R <	2R -	2R -	2R -	2R -
	3R -	3R <	3R -	3R -	3R -	3R -
	4R -	4R <	4R -	4R -	4R -	4R >
	5R -	5R <	5R -	5R -	5R >	5R >
	6R -	6R <	6R -	6R -	6R >	6R >

6.3 Increasing the Developer Count

Table 5 displays the result of a series of comparisons between projects with two developers against projects with 3, 4, and 5 developers respectively. We see very similar trends to the previous set of results in section 6.2.

7. CONCLUSIONS AND FUTURE WORK

Table 6 presents a summary of the discernable trends across each metric type as we add developers to a project team. We can see very similar trends regardless of whether we compare projects with a single developer against those with several developers, or whether we compare projects with multiple developers with those with still more developers.

Table 5 Results from the 5 developer against 2, 3, and 4 developer comparisons.

2D v 5D	CBO	DIT	LCOM	NOC	RFC	WMC
	17%	17%	0%	0%	17%	0%
	1R <	1R -	1R -	1R -	1R -	1R -
	2R -	2R -	2R -	2R -	2R -	2R -
	3R -	3R -	3R -	3R -	3R -	3R -
	4R -	4R <	4R -	4R -	4R -	4R -
	5R -	5R -	5R -	5R -	5R -	5R -
	6R -	6R -	6R -	6R -	6R >	6R -

3D v 5D	CBO	DIT	LCOM	NOC	RFC	WMC
	67%	50%	33%	0%	33%	50%
	1R <	1R -	1R -	1R -	1R <	1R <
	2R <	2R -	2R -	2R -	2R <	2R <
	3R -	3R <	3R >	3R -	3R -	3R -
	4R <	4R <	4R >	4R -	4R -	4R -
	5R -	5R <	5R -	5R -	5R -	5R -
	6R <	6R -	6R -	6R -	6R -	6R >

4D v 5D	CBO	DIT	LCOM	NOC	RFC	WMC
	67%	17%	67%	0%	50%	67%
	1R <	1R -	1R -	1R -	1R <	1R -
	2R >	2R -	2R -	2R -	2R >	2R -
	3R -	3R -	3R >	3R -	3R -	3R >
	4R <	4R <	4R >	4R -	4R -	4R >
	5R -	5R -	5R >	5R -	5R -	5R >
	6R <	6R -	6R >	6R -	6R >	6R >

Table 6. Summary of observed trends. Shaded boxes indicate a negative impact. Clear boxes indicate a positive impact.

Metrics	Objective	Lone developer v. Multi-developer teams	Smaller teams v. larger teams
CBO	↓	↓	↑
DIT	↓	↑	↑
LCOM	↓	↑	↑
NOC	↓	-	-
RFC	↓	↓	↓
WMC	↓	↓	↓

The clear conclusion is that where projects are collaborated on by a larger number of developers we are likely to see a decrease in cohesion (reflected by larger LCOM values), an increase in

structural complexity (reflected by larger DIT values) and an increase in coupling (reflected by larger CBO values). On a more positive note, we are likely to see that classes have improved functional decomposition (reflected by lower RFC and WMC values).

Finally, we can see consistency between our results and the research of both the work of Nagappan et al. [7] who linked larger team sizes with increased fault-proneness and Basili et al. confirmed that higher values of CBO and DIT, as observed in our research, is highly correlated with increased defect counts [9].

The implications of these findings are of relevance to practitioners. We believe that software development teams should take note of the fact that the structural attributes of a codebase can show degradation in some aspects as team sizes grow and the diligent use of tools like SonarQube [27] can provide visibility of this to developers and management in order for them to work together to mitigate any negative trends. This work is also relevant to the research community, to whom we suggest a number of avenues that we encourage the research community to pursue.

Firstly, we can hypothesize that the reasons driving the trends observed in table 6 are that, while a competent developer with relative unfamiliarity with the codebase will be capable of implementing code with a high degree of functional decomposition (for example, smaller single purpose methods), achieving low coupling and complexity as a codebase evolves typically requires a more in-depth understanding of the entirety of the codebase – an understanding that we can reasonably hypothesize is more likely to be lacking in a larger development team. Through qualitative analysis and engaging development team members, the research community could shed more light as to the drivers behind the trends revealed in this study.

Secondly, we believe that there is value in looking at how this work applies within the context of Agile development [22]. For example the Agile methodology recommends that development teams are co-located to facilitate communication between members [23]. In contrast, open source project teams tend to consist of people collaborating without necessarily sharing the same physical space. There is value in understanding if the negative effects of a larger development team manifest when a team is co-located and experiences lower barriers to effective communication.

Finally, we appreciate that a team with a total of 5 developers cannot necessarily be classed a large team – indeed this is considered the lower limit of the ideal Agile development team. We would be interested to see how these trends continue when taken up to and beyond the maximum ideal team size of 9 as stipulated by the Agile approach [23].

8. REFERENCES

[1] S. O'Grady, What Black Duck Can Tell Us About GitHub, Language Fragmentation and More, RedMonk, 2011, www.redmonk.com/sograde/2011/06/02/blackduck-webinar (accessed 15/05/2015).

[2] Tiobe Software, TIOBE programming community index for June 2013, 2013, www.tiobe.com (accessed 15/05/2015)

[3] C. Jones, O. Bonsignour, The economics of software quality. Addison-Wesley Professional, 2011.

[4] S. Kan, Software Quality Metrics Overview. Metrics and Models in Software Quality Engineering, 2002, pp. 85-120.

[5] F. Brooks, The Mythical Man-Month. Addison-Wesley, 1975.

[6] J. Rodger, P. Pankaj, A. Nahouraii, Knowledge Management of Software Productivity and Development Time. Journal of Software Engineering and Applications, 4(11), 2011, pp. 609.

[7] N. Nachiappan, B. Murphy, V. Basili, The Influence of Organizational Structure on Software Quality: An Empirical Case Study. Proceedings of the 30th international conference on Software engineering, 2008.

[8] R. Prather, An Axiomatic Theory of Software Complexity Measure. The Computer Journal, 27(4), 1984, pp. 340-347.

[9] V. Basili, R., & L. Briand, W. Melo, A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering, 22(10), 1996, pp. 751-761.

[10] R. Subramanyam, M. Krishnan, Empirical Analysis of CK Metrics For Object-Oriented Design Complexity: Implications For Software Defects. IEEE Transactions on Software Engineering, 29(4), 2003, pp 297-310.

[11] K. El Emam, W. Melo, J. Machado, The Prediction of Faulty Classes Using Object-Oriented Design Metrics. Journal of Systems and Software, 56(1), 2001, pp 63-75.

[12] M. Tang, M. Kao, M. Chen, An Empirical Study on Object-Oriented Metrics. Proceedings of the Sixth International Software Metrics Symposium, 1999, pp. 242-249.

[13] J. Xu, D. Ho, L. Capretz, An Empirical Validation of Object-Oriented Design Metrics For Fault Prediction. Journal of Computer Science, 4(7), 2008, pp 571.

[14] R. Malhotra, A. Jain. Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality. Journal of Information Processing Systems, 8(2), 2012, pp 241-262.

[15] H. Saberwal, S. Singh, S. Kaur. Empirical Analysis Of Open Source System For Predicting Smelly Classes. International Journal of Engineering Research & Technology, 2(3), 2013.

[16] L. Badri, M. Badri, F. Toure, An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes. International Journal of Software Engineering and its Applications, 5(2), 2011, pp. 69-85.

[17] S. Chidamber, C. Kemerer, A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20(6), 1994, pp. 476-493.

[18] W. Li, S. Henry, Object-Oriented Metrics That Predict Maintainability. Journal of Systems and Software, 23(2), 1993, pp. 111-122.

[19] F. Akiyama, An Example of Software System Debugging. IFIP Congress 71(1), 1971.

[20] J. Howison, M. Conklin, K. Crowston, FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. International Journal of Information Technology and Web Engineering, 1(3), 2006, pp. 17–26.

[21] G. Robles, S. Koch, J, González-Barahona, J. Carlos, Remote Analysis and Measurement of Libre Software Systems by Means of the CVSAnalY tool. Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software System, 2004, pp. 51-55.

[22] L. Lindstrom, R. Jeffries, Extreme Programming and Agile Software Development Methodologies. Information Systems Management, 2005, 21(13).
[23] K. Schwaber, J. Sutherland, The Scrum Guide. Scrum.org, 2014, www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf (accessed 15/05/2015).
[24] R. Smith, J. Hale, A. Parrish, An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation. IEEE Transactions on Software Engineering, 27(3), 2001, pp. 264-271.
[25] E. Capra, A. Wasserman, A Framework for Evaluating Managerial Styles in Open Source Projects. Open Source Development, Communities and Quality, 2008, pp. 1-14.
[26] The International Software Benchmarking Standards, www.isbsg.org (accessed 15/05/2015).
[27] SonarQube, www.sonarqube.org (accessed 15/05/2015).
[28] SciTools, www.scitools.com (accessed 15/05/2015).