

The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules

A. Güneş Koru, *University of Maryland, Baltimore County*

Khaled El Emam, *University of Ottawa*

Our observations on large-scale software systems lead to an empirically based theory that smaller modules will be proportionally more dependent compared to larger ones.

Our recent research on several large-scale software products has consistently shown that smaller modules are *proportionally* more defect prone.¹⁻³ These findings challenge the common recommendations from the literature suggesting that quality assurance (QA) and quality control (QC) resources should focus on larger modules. Those recommendations are based on the unfounded assumption that a monotonically increasing linear relationship exists between module size and defects.¹⁻⁴ Given that complexity is correlated with size,⁵ following

such recommendations ensures that the more complex modules receive greater scrutiny. However, our recent findings imply that, given limited and a fixed amount of resources, focusing on the smaller modules would lead to more effective defect detection.¹⁻³ So, it's important to understand the mechanisms behind those findings to make the case for amending current practices.

Research has shown that higher module dependencies, also called higher *coupling*, result in more defects by increasing the likelihood of interface defects.⁶⁻¹⁰ "Coupling" as we use it here corresponds to the concept of *fan-out* (depending on other modules). It doesn't mean *fan-in* (being depended upon by other modules) because the literature shows no consistent evidence that a module's fan-in is related to its defect-proneness. Relying on the evidence from the literature, we

formulated a testable hypothesis of relative dependency (RD) for software modules:

H_{RD} : *Smaller modules are proportionally more coupled.*

H_{RD} requires us to study the size-coupling relationship for software modules. So far, the empirical evidence has shown a positive correlation between module size and coupling because of their monotonically increasing relationship.¹¹⁻¹³ However, those findings aren't enough to test H_{RD} because correlations don't imply proportionality between the two variables.

To investigate proportionality, we performed an empirical study of data from multiple systems. The results confirmed H_{RD} and led us to formulate the *theory of relative dependency*:

Table 1

**The analyzed products, their functionality,
and descriptive statistics for their size and coupling data**

Product	Functionality	No. of classes	LOC				Coupling between object classes (CBO)*				Depth of inheritance tree (DIT)*			
			Min.	Median	Max.	Total	Min.	Median	Max.	Total	Min.	Median	Max.	Total
Mozilla	Web browser	4,971	1	59.0	12,408	990,571	0	4.0	157	37,210	0	1	9	8,393
KWord	Word processing	224	6	66.0	6,634	44,473	0	7.0	185	2,239	0	1	3	237
KSpread	Spreadsheet	284	1	65.5	5,159	67,508	0	5.5	103	2,368	0	1	3	252
KPre-senter	Presentation	178	9	89.5	5,884	47,204	0	5.0	150	1,666	0	1	4	212
Kexi	Data management	628	4	54.0	3,460	80,795	0	5.0	83	4,787	0	1	4	621
KPlato	Project management	218	7	41.0	1,354	23,884	0	4.0	68	1,337	0	1	3	277
Kivio	Diagramming	130	5	75.5	1,955	23,721	0	5.0	84	1,049	0	1	3	131
Krita	Painting and image editing	700	3	43.0	2,965	74,604	0	4.0	126	4,315	0	1	5	1,025
KChart	Chart drawing	85	9	144.0	6,700	31,368	0	7.0	28	612	0	1	3	89
Karbon	Scalable-vector drawing	183	9	92.0	1,293	26,468	1	7.0	81	1,588	0	1	3	247
KFilters	Conversion between file formats	954	1	50.5	2,151	133,688	0	2.0	98	3,964	0	1	5	683
KO. Library	KOffice library	699	3	62.0	2,716	104,609	0	4.0	56	4,569	0	1	8	793
Kugar	Generating business-quality reports	55	15	52.0	847	6,765	0	3.0	29	298	0	1	5	108
KFormula	Formula editor for KOffice	17	8	38.0	404	1,337	0	0	15	50	0	1	2	19

*CBO indicates the number of other classes whose methods or attributes are used; DIT indicates the maximum depth of a class in the inheritance tree from the root class.

In large-scale software systems, smaller modules will be proportionally more dependent compared to larger ones.

This theory, along with our findings, has important implications for software practice, especially for projects involving considerable refactoring.

Methods

We performed measurements and tested H_{RD} on 14 open source object-oriented (OO) products from different development teams: the Mozilla 1.0 Web browser and 13 products in KOffice 1.6.3, an office suite. The release dates for Mozilla 1.0

and KOffice 1.6.3 were 5 June 2002 and 7 June 2007, respectively. Table 1 lists the products and their functionality; the list of products covers a wide range of functionalities and measurement characteristics.

Data Collection

We measured size in physical LOC excluding blank lines and lines including only comments. To measure coupling, we used two OO metrics defined by Shyam Chidamber and his colleagues:¹⁴

- *Coupling between object classes (CBO)* is the number of other classes whose methods or attributes are used.

■ *Depth of inheritance tree* (DIT) is the maximum depth of a class in the inheritance tree from the root class.

We considered only application (product) classes, not those in the C++ libraries. Empirical studies of software engineering have commonly used all three metrics.

In OO programming, classes are considered to be logically cohesive development units, or modules, organized in an inheritance hierarchy. So, we must consider DIT to better understand class dependencies. Indeed, Chidamber and his colleagues stated that DIT is “a measure of how many ancestor classes can potentially affect this class,” which exactly describes the dependencies of a child class to its ancestors.

We performed the measurements at the class level using a mature static-code-analysis tool, Understand for C++ (www.scitools.com). For each class, we took into account its header and implementation files during the size measurements. As a result of our measurements, we created a distinct data set for each product, with the data points corresponding to the C++ classes in the product. Each data point included three numeric values for LOC, CBO, and DIT.

Table 1 includes the descriptive statistics for the data sets for the products in this study.

Data Analysis

For each product, we plotted a *concentration curve* and produced a *concentration index* (C) to visualize and quantify the inequality of CBO concentration with respect to module size. We repeated this process for DIT. For more on concentration curves and indices, see the “Using Concentration Curves and Indices” sidebar.

Results

Figure 1 displays the concentration curves for all the products. All the curves except the CBO curve for KFormula are above the equality line, clearly showing that coupling is concentrated in smaller modules. For example, in KChart, the smallest module sizes adding up to 20 percent of the total size had more than 50 percent of the total CBO. In all plots, the inequalities with respect to size were even greater for DIT. For example, in KSpread, the source lines in the smallest modules adding up to 20 percent of the total size had almost 80 percent of the total DIT.

Figure 2 shows bar plots for the concentration indices. The estimate of C is the value on the y -axis corresponding to the bar's center. A

bar's length shows the 95 percent confidence interval. For ease of reference, the figure also shows the numeric values of C and its confidence intervals.

Figure 2 shows that, for all products except KFormula, not only the point estimates of C but also the 95 percent confidence intervals stay above the zero line (the dashed horizontal line), showing statistical significance. For KFormula, the point estimate for C is also positive; however, its confidence interval reaches below zero. This is understandable because KFormula is a small product providing only 17 data points, which makes it difficult to reject the equality of CBO concentration ($C = 0$) with enough confidence.

Figure 2 presents compelling quantitative evidence that, in general, coupling is concentrated more in the lines of source code in smaller modules. So, smaller modules are proportionally more coupled than larger modules. This strongly supports H_{RD} .

How Code Refactoring Affects Coupling Concentration

Refactoring basically means improving software design without changing its functionality.¹⁵ Martin Fowler and his colleagues stress the necessity of refactoring because software design typically decays over time as developers add or change functionality.¹⁵ So, refactoring is inherent in software design and maintenance and is particularly favored in agile development.

When discussing “code smells,” Fowler and his colleagues stress that the first three smells in the “stink parade” are *duplicated code*, *long method*, and *large class*.¹⁵ So, the expert advice directly suggests focusing refactoring first (or more) on larger modules. Some of the other code smells such as *feature envy* and *shotgun surgery* recommend focusing on tightly coupled classes to reduce their dependency, which is in concordance with the traditional advice given to practitioners. Yoshio Kataoka and his colleagues reported that experts' subjective judgment on refactoring's effectiveness correlated with coupling reductions.¹⁶

Indirectly, more emphasis on highly coupled modules also means more emphasis on larger modules, for two reasons:

- As we mentioned before, a well-established correlation exists between size and coupling.
- Mika Mäntylä and his colleagues examined the correlations between code smells.¹⁷ They

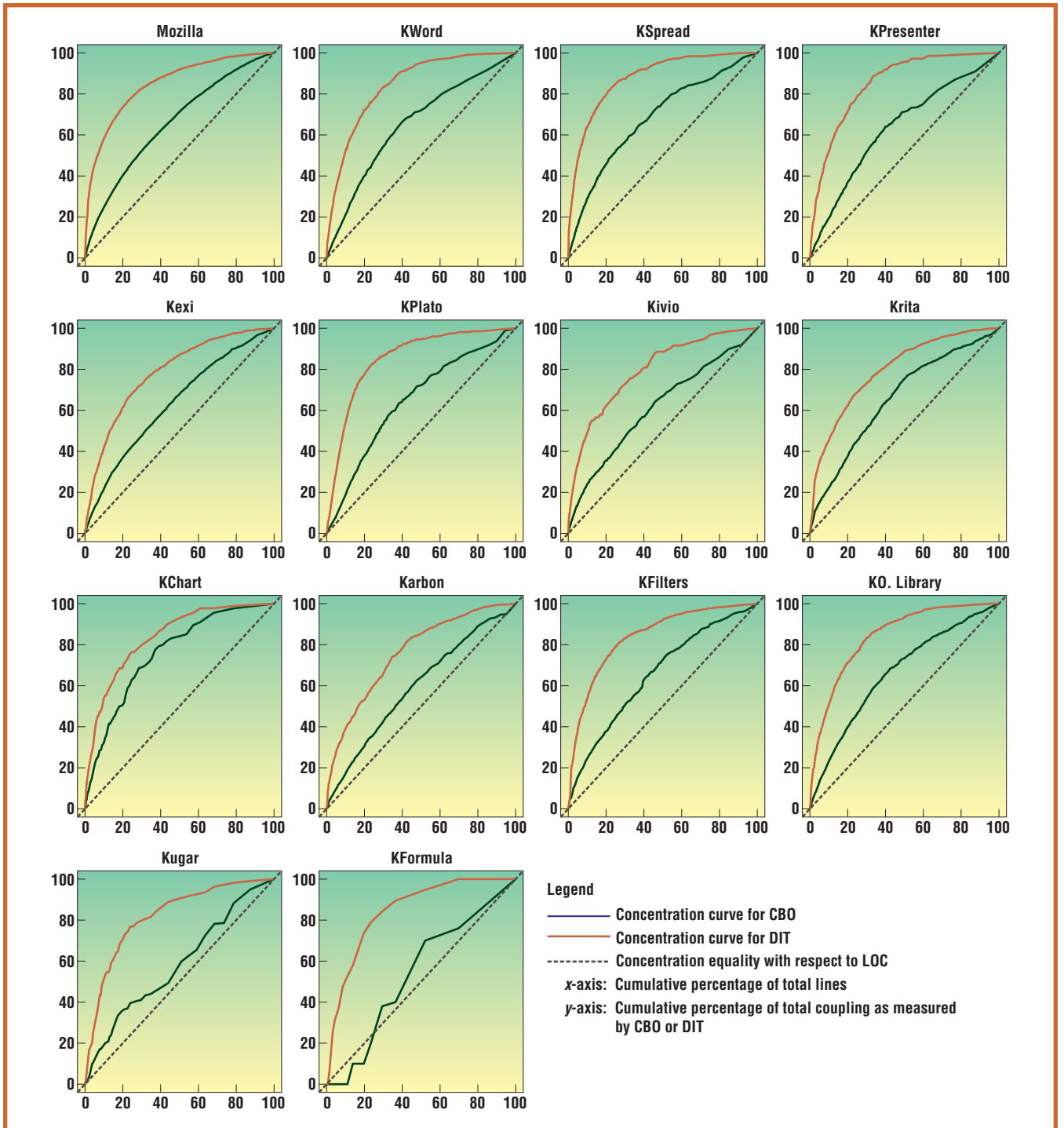


Figure 1. Concentration curves for coupling between object classes (CBO) and depth of inheritance tree (DIT), for all products tested. The x- and y-axes represent the cumulative proportion of total LOC and CBO or DIT, respectively. The curves are plotted for unique class sizes sorted from the smallest to the largest and marked along the x-axis from left to right. All the curves except KFormula's CBO curve are above the equality line, showing that coupling is concentrated in smaller modules.

found that the Spearman correlation between large class (which is concerned with size) and feature envy (which is concerned with coupling) was 0.59 and significant at $\alpha = 0.01$.

We also examined whether refactoring tools' smell detection rules detect larger classes. Borland Together (www.borland.com/us/products/together), a Java IDE, includes a refactoring tool.

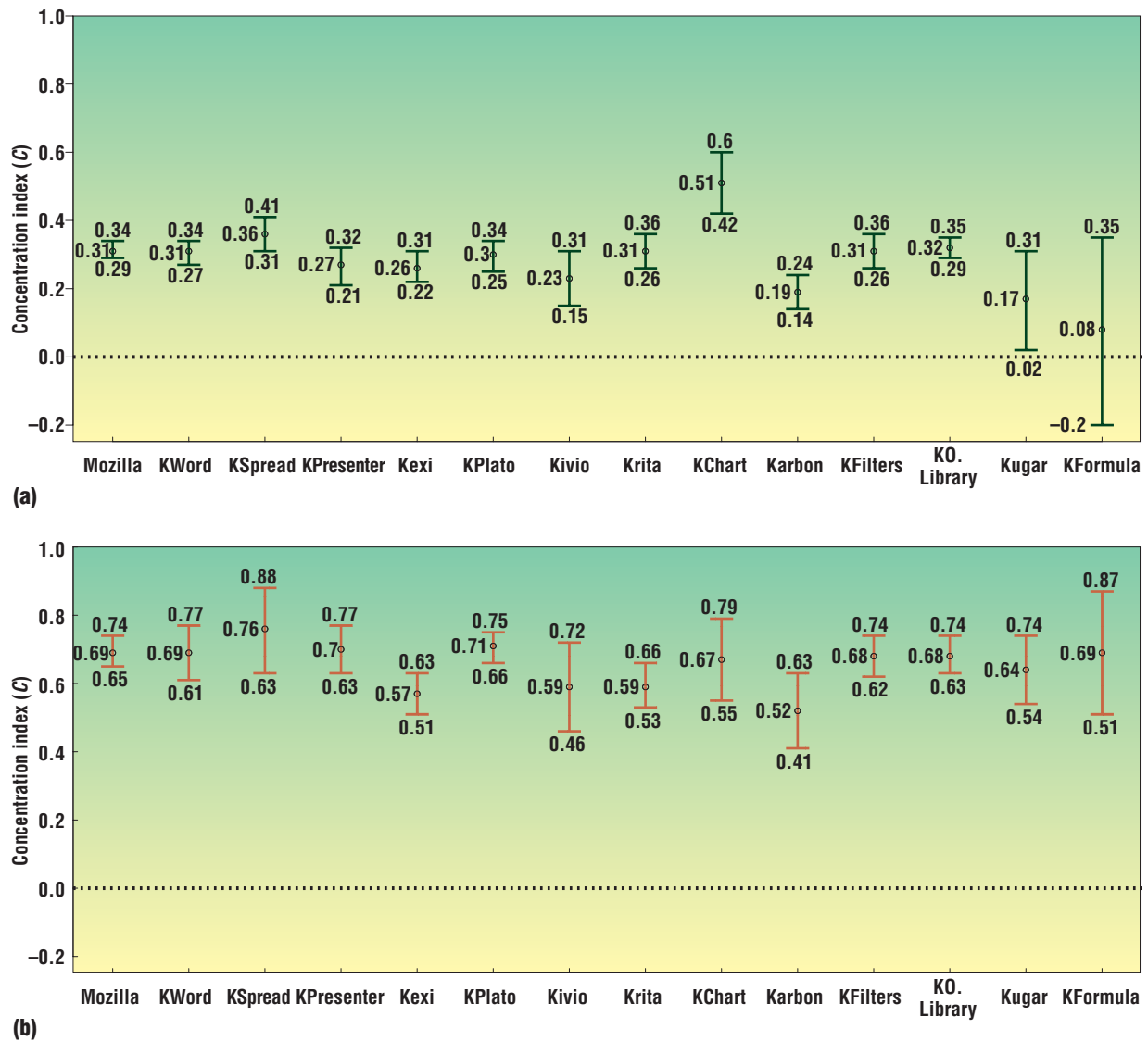


Figure 2. Coupling concentration index (C) values with their confidence intervals for all products studied. (a) CBO and (b) DIT. $C > 0$ means that smaller modules are proportionally more coupled. The results indicate that, in general, coupling is concentrated more in the lines of source code in smaller modules.

Because the tool supports smell detection for only Java programs, we detected the code smells in JBoss (www.jboss.org), an open source Java application server. (JBoss isn't in Table 1 because Understand for C++ analyzed C++ source code.) Borland Together detected 7,991 classes in JBoss.

Table 2 shows that, for each smell other than *refused bequest*, a statistically significant size difference existed between the classes having the smell and those not having it. For the rest of the code smells, the percentiles for median size rankings were very high, directing developers' attention to larger modules.

All this evidence suggests that larger classes will more likely be refactored.

Fortunately, we were also able to investigate the effect of refactoring on coupling concentration empirically. In the Kivio and Karbon projects, the developers were specifically requested to perform refactoring before an upcoming big release, KOffice 2.0. (We archived this announcement at www.webcitation.org/5SrjtxRoj.) At the time of writing, the fourth alpha release of KOffice 2.0 had occurred. When we examined the Kivio sources for this release and compared them with the 1.6.3 release, we saw that Kivio was almost unchanged, showing no signs of refactoring. However, for Karbon, we observed that refactoring had indeed taken place in the alpha release.

The Karbon 1.6.3 release (before refactoring)

Using Concentration Curves and Indices

While investigating the inequality of coupling with respect to size, we adopted concentration curves and indices. This approach is widely used in healthcare and public health to measure a health outcome's inequality with respect to a metric indicating socioeconomic status.¹

Usually, plotting a concentration curve involves plotting the cumulative proportion of the ill-health variable (for example, infant mortality or the number of heart attacks) on the y-axis against the cumulative proportion of the population on the x-axis, ordered from lowest to highest socioeconomic status (for example, income or education). Figure A shows such a typical curve, using an ill-health variable.

If there's no inequality of ill health related to socioeconomic status, the concentration curve follows the 45-degree line (diagonal), which is called the *equality line*. If ill health is concentrated in the disadvantaged socioeconomic levels, the concentration curve rises above the equality line as in Figure A. If the advantaged socioeconomic levels had a higher concentration of ill health, the curve would drop below the equality line.

The ordering along the x-axis can be at the individual or group level.¹ The groups usually

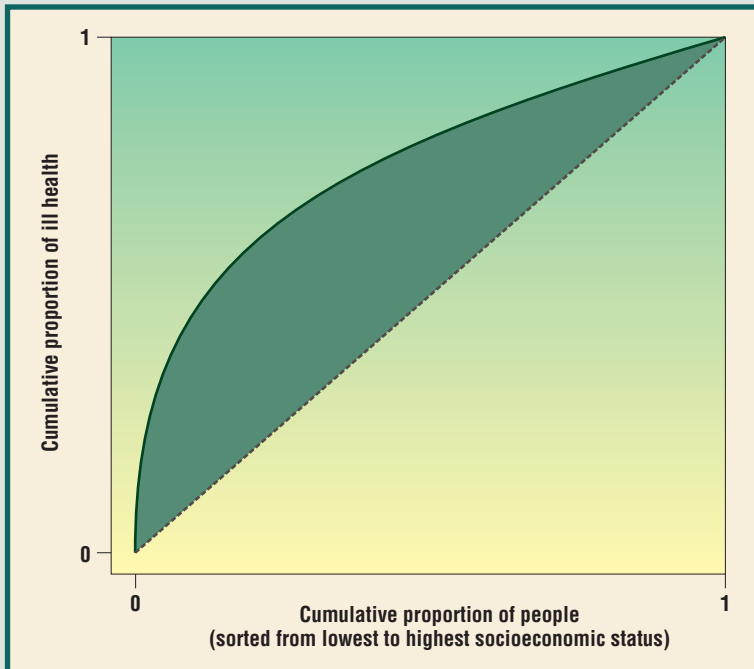


Figure A. A typical concentration curve used in health research. In this case, the curve indicates a higher concentration of ill health in disadvantaged socioeconomic levels.

Table 2

Code smells in JBoss as determined by Borland Together, with data about classes having the smells*

Code smell	Number of classes with this smell	Statistical size comparison with classes not having the smell (p-value from the Wilcoxon rank-sum test)	Median size ranking for classes having this smell (% , with 100% being the largest size)
Duplicated code	64	<2.2e-16	95.95
Long method	226	<2.2e-16	95.84
Large class	59	<2.2e-16	98.60
Shotgun surgery	199	<2.2e-16	84.88
Feature envy	48	<2.2e-16	94.14
Message chains	90	4.441e-16	80.23
Refused bequest	221	0.60	49.59
Data class	25	6.286e-06	74.57

*We obtained the median ranking after assigning a size rank to each class (we computed an average rank value for equal rankings) and then finding the median-rank value for the sizes having the code smell.

had 183 classes with 26,468 total LOC, with a median class size of 92 LOC. Ninety-six classes were changed, 48 were added, and 66 were deleted. After refactoring, Karbon had 163 classes with 20,617 total LOC, with a median class size

of 81 LOC. These numbers show that refactoring affected much of the system. In the resulting alpha version, the newly added classes were significantly smaller than the others (the Wilcoxon rank-sum test gives $p = 0.005$). No size difference existed be-

correspond to some welfare categories—for example, the income categories for neighborhoods described as very poor, poor, middle-class, rich, and very rich. In the grouped case, the group sums are used to calculate the proportions of people and ill health.

By definition, the concentration index (denoted with C), which is the measure of inequality, is twice the area between the concentration curve and the equality line. This area is the shaded part of Figure A. The socioeconomic-related inequality of ill health increases as the shaded area increases. C becomes zero when the concentration curve exactly overlaps with the equality line. It can take any value between +1 and -1. In our research, when the curve rises above the equality line, C increases toward +1; it decreases toward -1 when the curve drops below the equality line. (In health-related fields, C usually decreases when the curve rises above the equality line because ill health for the disadvantaged [or poor] is a more common and undesirable phenomenon. We followed the opposite of this convention to simplify the presentation in the main article.)

For the products in our study, we plotted the cumulative proportion of total coupling (on the y -axis) versus the cumulative proportion of total lines of source code (on the x -axis) grouped and ordered by class size. That is, we adopted the grouped case we just explained and used class size (in a way similar to the welfare categories) to group lines of source code. In this scheme, each unique class size formed

a size category. Then, we ordered the size categories from the smallest to largest and calculated the cumulative proportions for each category to draw the concentration curves and to calculate C . We calculated the variance and confidence interval for C using the method detailed by Nanak Kakwani and his colleagues.¹

When this process is followed, if our hypothesis is true, coupling should be concentrated in the source lines in smaller modules, and the concentration curves obtained should resemble the one in Figure A rising above the equality line. Such a curve's C value should be positive, and, if the concentration effect is significant, the 95 percent confidence interval for C should remain above zero.

To summarize, the concentration curve lets us visualize the nonlinear relationship between module size and coupling. The concentration index, C , quantitatively summarizes the visual information that curve presents. The sign of C indicates the coupling concentration's direction (that is, positive when coupling is concentrated in the source code lines in smaller modules, and vice versa). Its absolute value shows the concentration's strength; its statistical significance is important and should be calculated.

Reference

1. N. Kakwani, A. Wagstaff, and E. Van Doorslaer, "Socioeconomic Inequalities in Health: Measurement, Computation, and Statistical Inference," *J. Econometrics*, vol. 77, no. 1, 1997, pp. 87–103.

tween the removed and nonremoved classes.

Refactoring didn't affect DIT values significantly. However, among the modified classes, CBO increased proportionally more for smaller classes. The CBO ratio (CBO after refactoring divided by CBO before refactoring) decreased as class size increased, giving a Spearman's correlation of -0.32, which was highly significant ($p = 0.002$). The CBO concentration index was 0.18 before refactoring and 0.23 after it, showing that coupling was concentrated even more in smaller classes after refactoring.

So, overall, the Karbon analysis shows that refactoring

- produced more classes that were smaller and
- disproportionately increased coupling for smaller classes.

Refactoring activities seem likely to increase the concentration of coupling in smaller modules. Given that refactoring is continuous, especially for agile development projects,¹⁸ this result suggests that systems developed with agile methods

will likely face an increasing concentration of coupling in smaller modules. So, giving a higher priority to smaller modules for defect detection will be even more important. As always, in such decisions, developers should also consider other available information about software modules such as their fan-in, business importance, and operational profile, as we stressed in our earlier research.^{1–3}

Limitations

As with any empirical study, our study has limitations. Certainly, it is possible to use or derive some metrics other than LOC, CBO, and DIT. However, software researchers have widely adopted, validated, and accepted these metrics. Future studies could collect additional measures of size and coupling to compare their results with ours.

Our results are limited to the concentration of dependencies over source code lines with respect to module size. Future studies could also investigate the effect of refactoring on the size-coupling relationship in more depth and whether some of

Basili and Perricone's Work on Module Size and Defect Density

Victor Basili and Barry Perricone studied defect density for different size groups and observed higher defect density for smaller modules.¹ However, their analysis suffered from artificial ratio correlations.^{2,3} When defect density is plotted against size or tabulated against size groups, even randomly generated data will show higher defect density for smaller modules because size is in the denominator of the derived metric, defect density.


Nevertheless, Basili and Perricone concluded that smaller modules are proportionally more problematic; our recent findings⁴⁻⁶ concur with their observation. Then, they speculated that the equal distribution of interface defects over smaller and larger modules could have been responsible for their finding, which implies that smaller modules had proportionally more interface defects. Their definition of interface defects was "those (defects) that were associated with structures existing outside the module's local environment but

which the module used."¹ This definition exactly describes the defects associated with coupling. So, our findings in the main article about a higher concentration of coupling in smaller modules shows that Basili and Perricone's speculation was plausible.

References

1. V.R. Basili and B.T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Comm. ACM*, vol. 27, no. 1, 1984, pp. 42-52.
2. K. El Emam et al., "The Optimal Class Size for Object-Oriented Software," *IEEE Trans. Software Eng.*, vol. 28, no. 5, 2002, pp. 494-509.
3. J. Rosenberg, "Some Misconceptions about Lines of Code," *Proc. 4th Int'l Symp. Software Metrics (Metrics 97)*, IEEE CS Press, 1997, pp. 137-142.
4. A.G. Koru et al., "Theory of Relative Defect Proneness," *Empirical Software Eng.*, vol. 13, no. 5, 2008, pp. 473-498.
5. A.G. Koru et al., "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," *IEEE Trans. Software Eng.*, vol. 35, no. 2, 2009, pp. 293-304.
6. A.G. Koru et al., "Testing the Theory of Relative Defect Proneness for Closed-Source Software," to be published in *Empirical Software Eng.*

the process and people characteristics (developer skill or experience, testing or inspection effort, and so on) have any effect on this relationship.

the software field's knowledge and capacity to develop and maintain high-quality software systems effectively and efficiently. 

Our study has two implications for practice. First, our findings provide a plausible mechanism to explain why smaller modules have proportionally more defects as observed in our earlier studies and in Victor Basili and Barry Perricone's research (see the "Basili and Perricone's Work on Module Size and Defect Density" sidebar). Coupling is concentrated more in the smaller modules, making them proportionally more defect-prone. This evidence will be useful to practitioners when they seek resources or support to amend or revise their organizations' QA and QC practices.

Second, refactoring exacerbates the coupling concentration in smaller modules. So, for projects that refactor extensively, focusing defect detection and correction on smaller modules will increase the QA and QC effectiveness even more.

Certainly, we don't claim that software developers should avoid refactoring. Refactoring is essential for software development and maintenance because of its system-wide benefits. Our findings are only about the effect of refactoring on coupling concentration with respect to module size, which has implications in terms of the effectiveness and efficiency of focused defect detection activities often performed with limited resources.

Empirical studies investigating our findings and their implications will further contribute to

Acknowledgments

We thank the associate editor and the anonymous reviewers for their useful and constructive feedback. We also thank Vic Basili, Dave Card, Barbara Kitchenham, Tim Menzies, and Carolyn Seaman for their feedback on earlier drafts of this article.

References

1. A.G. Koru et al., "Theory of Relative Defect Proneness," *Empirical Software Eng.*, vol. 13, no. 5, 2008, pp. 473-498.
2. A.G. Koru et al., "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," *IEEE Trans. Software Eng.*, vol. 35, no. 2, 2009, pp. 293-304.
3. A.G. Koru et al., "Testing the Theory of Relative Defect Proneness for Closed-Source Software," to be published in *Empirical Software Eng.*
4. K. El Emam et al., "The Optimal Class Size for Object-Oriented Software," *IEEE Trans. Software Eng.*, vol. 28, no. 5, 2002, pp. 494-509.
5. N. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed., PWS Publishing, 1996.
6. J.H. Hayes et al., "Fault Links: Exploring the Relationship between Module and Fault Types," *Dependable Computing—EDCC 2005*, LNCS 3463, Springer, 2005, pp. 415-434.
7. D.E. Perry, "Programmer Productivity in the Inscape Environment," *Proc. IEEE Global Communications Conf. (GlobeCom 86)*, IEEE Press, 1986, pp. 428-434.
8. D.E. Perry and W.M. Evangelist, "An Empirical Study of Software Interface Errors," *Proc. IEEE Int'l Symp. New Directions in Computing*, IEEE Press, 1985, pp. 32-38.
9. D.E. Perry and W.M. Evangelist, "An Empirical Study of Software Interface Faults—an Update," *Proc. 20th Ann. Hawaii Int'l Conf. Systems Sciences (HICSS 20)*, IEEE Press, 1987, pp. 113-126.

10. D.E. Perry and C.S. Stieg, "Software Faults in Evolving a Large, Real-Time System: A Case Study," *Proc. 4th European Conf. Software Eng.*, Springer, 1993, pp. 48–67.
11. L.C. Briand et al., "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems," *J. Systems and Software*, vol. 51, no. 3, 2000, pp. 245–273.
12. K. El Emam et al., "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Trans. Software Eng.*, vol. 27, no. 7, 2001, pp. 630–650.
13. T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.*, vol. 31, no. 10, 2005, pp. 897–910.
14. S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Eng.*, vol. 248, no. 1998, pp. 629–639.
15. M. Fowler et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999.
16. Y. Kataoka et al., "A Quantitative Evaluation of Maintainability Enhancement by Refactoring," *Proc. IEEE Int'l Conf. Software Maintenance*, IEEE CS Press, 2002, pp. 576–585.
17. M. Mäntylä, J. Vanhanen, and C. Lassenius, "A Taxonomy and an Initial Empirical Study of Bad Smells in Code," *Proc. IEEE Int'l Conf. Software Maintenance*, IEEE CS Press, 2003, pp. 381–384.
18. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

About the Authors



Khaled El Emam is an associate professor at the University of Ottawa's Faculty of Medicine and School of Information Technology and Engineering. He is a Canada Research Chair in Electronic Health Information at the university. El Emam has a PhD from the Department of Electrical and Electronics Engineering, King's College, University of London. Contact him at kelemam@uottawa.ca; www.ehealthinformation.ca.

A. Güneş Koru is an assistant professor in the Department of Information Systems at the University of Maryland, Baltimore County. His research interests include software quality, measurement, maintenance, and evolution; open source software; bioinformatics; and healthcare informatics. Koru has a PhD in computer science from Southern Methodist University. He's the program chair for the Promise (Predictive Models in Software Engineering) 2010 conference. Contact him at gkoru@umbc.edu; <http://umbc.edu/~gkoru>.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

ADVERTISER INFORMATION

MARCH/APRIL 2010 • IEEE SOFTWARE

Advertiser

ICSE 2010	Page 32
John Wiley & Sons	1
Nu Info Systems, Inc.	8
Saturn 2010	Cover 2
Seapine Software	Cover 4
XP 2010	7

Advertising Personnel

Marion Delaney
IEEE Media, Advertising Dir.
Phone: +1 415 863 4717
Email: md.ieeemedia@ieee.org

Marian Anderson
Sr. Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
Sr. Business Development Mgr.
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Recruitment:

Mid Atlantic
Lisa Rinaldo
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: lr.ieeemedia@ieee.org

New England
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Southeast
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynnntom@mindspring.com

Midwest/Southwest
Darcy Giovino
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

Northwest/Southern CA
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Japan
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe
Heleen Vodegel
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com

Product:

US East
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: db.ieeemedia@ieee.org

US Central
Darcy Giovino
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

US West
Lynne Stickrod
Phone: +1 415 931 9782
Fax: +1 415 931 9782
Email: ls.ieeemedia@ieee.org

Europe
Sven Anacker
Phone: +49 202 27169 11
Fax: +49 202 27169 20
Email: sanacker@intermediapartners.de