Does an 80:20 rule apply to Java coupling?

Asma Mubarak, Steve Counsell, and Robert M. Hierons,
Department of Information Systems and Computing,
Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
{asma.mubarak, steve.counsell, rob.hierons}@brunel.ac.uk

Objective. To explore whether an 80:20 rule exists in Java from six coupling metrics over multiple versions of open-source software and, if so, whether that relationship is exacerbated over time.
Methods. We used the automated tool JHawk to extract the 6 different coupling metrics from four Open-Source Systems. We then ranked the classes on each of these 6 coupling metrics and then analysed the top 20% of classes to see whether 80% of total coupling was contained therein.
Conclusions. Only one metric appeared consistently to have an 80:20 relationship and that was the 'fan-in' metric. Evidence suggests that fan-in and fan-out have a complementary relationship. We found many of the other metrics had few, if any such relationships. The RFC was typical in this sense – no 80:20 relationship was found in any of the systems or any version in those systems. We also found no evidence to support the view that over time, the 80:20 is exacerbated.

Keywords: Power Law, open source systems, coupling, evolution.

1. INTRODUCTION

Many social and naturally occurring phenomena are distributed according to an 80:20 rule (sometimes known as a power law). In other words, 'small' occurrences of an artefact or phenomenon are extremely common whereas 'large' instances are relatively rare. Wheeldon and Counsell [9] illustrated that a power law distribution existed in OO class relationships, particularly those related to coupling (via inheritance and aggregation). In this paper, we attempt to support or refute that earlier work by focusing on 6 separate, yet different coupling metrics to explore whether evolutionary coupling obeyed an 80:20 rule. We used an automated tool to extract each of the coupling metrics from four open-source systems (OSS). Excessive class coupling has often been related to the tendency for faults in software [4]. It is widely believed in the OO software engineering community that excessive coupling between classes creates a level of complexity that can complicate subsequent maintenance and potentially lead to the seeding of faults. In theory, we would expect coupling to increase through consistent application of maintenance as a system evolves and, hence, for any 80:20 rule to become exacerbated.

The research described in this paper relates to both software evolution and coupling [2, 3, 4]. In terms of software evolution, the laws proposed by Belady and Lehman [2] have provided the basis for many evolutionary studies in the past. In terms of coupling, a framework for its measurement was introduced in [3]; variations for different programming styles have also been proposed [1]. The role of method invocation in creating faults is also highlighted by Briand et al. [5]. Chidamber and Kemerer (C&K) proposed six OO metrics amongst which was the Response for a Class (RFC) metric used in this study [4]. The study presented also attempts to shed light on which coupling *types* tend to exhibit specific trends (in this case an 80:20 rule). There are also parallels with the use and credibility of 'key' classes, i.e., certain classes in any system that comprise a large number of methods (and, by implication, a large amount of coupling).

2. PRELIMINARIES

The four systems studied were:

1. Jasmin. A Java assembler which takes ASCII descriptions of Java classes and converts them into binary Java class files suitable for loading into a Java Virtual Machine. The system comprised 5 versions. It started with 5 packages and 110 classes and 5 packages and 130 classes by the latest version.
2. SmallSQL. A Java DBMS for Java desktop applications. It has a JDBC 3.0 interface and offers many ANSI SQL 92 and ANSI SQL 99 features. The system comprised 8 versions. It started with 130 classes in the first version and had 177 classes in the latest version.

3. DjVu. Provides an applet and desktop viewer Java virtual machine. Comprised 8 versions. It started with 12 packages and 77 classes in the first version with 14 packages and 79 classes in the latest version.
4. pBeans. Provides automatic object/relational mapping (ORM) of Java objects to DB tables. Comprised 10 versions, 4 packages and 36 classes in version 1 (10 packages and 69 classes in the latest version).

For each of the five systems, we collected six independent, coupling metrics using JHawk [6]:

1. Response for a Class (RFC): Defined by C&K [5]. The RFC is defined as the set of methods that can potentially be executed in response to a message received by an object of that class.
2. Number of EXTernal methods called (EXT): The more external methods that a class calls the more tightly bound that class is to other classes
3. Message Passing Coupling (MPC): The number of messages passed among objects of a class.
4. PACK. Number of imported PACKages [8].
5. Fan-in (FIN). The FIN of a function is the number of unique functions that call the function.
6. Fan-out (FOUT). Number of unique functions that a function calls.

3. DATA ANALYSIS

In the following analysis and for succinctness, we include in the tables for each system *only* the rows where at least one 80:20 rule was found to apply; equally, only the columns (i.e., metrics) where at least one 80:20 rule was found to apply are listed. If a value is omitted from a table, then no 80:20 rule applied in that case. (We note that for each system, the top 20% of classes will be *exactly* 20% of the total number of classes stated earlier in the description of the systems.) An 80:20 rule applies *iff* at least 80% of the coupling is incorporated in that top 20%.

3.1. The Jasmin System

Table 1 shows the percentage in each of the coupling metrics over the five versions (V1-V5) of the Jasmin system for 20% of the classes on a package basis (the 2 packages in this case are jas and jasmin). The most striking feature of the values in Table 1 is the absence of four of the six metrics extracted by the tool and subsequently analysed. No entries for RFC, EXT, PACK or FOUT greater than or equal to the threshold 80% were found.

Package	Version	MPC	FIN
jas	V1		90.15
	V2		89.35
	V3		83.41
	V4		84.78
	V5		84.78
jasmin	V1		90.71
	V2		92.9
	V3	81.39	94.19
	V4		92.9
	V5		93.33

TABLE 1: 80:20 metrics for the Jasmin system

In V1 of the jas package, the top 20% of the classes comprised over 90% of all FIN coupling. Comparing V1 and V5 shows that the 80:20 rule became weaker by V5 (i.e., at 84.78%). For the jasmin package, there is only a marginal increase in the 80:20 rule (from 90.71% to 93.33%). It is also worth noting that the FOUT metric had many values between 70% and 75% over the course of these versions and consequently are not shown in the table. Equally, the values of the other four metrics tended to be in the range 45%-70%, considerably lower than the FIN values.

3.2. The SmallSQL system

Table 2 shows the same data for the SmallSQL system. In common with the Jasmin system, the FIN metric satisfied the 80:20 rule across all versions studied. However, the rule is only marginally strengthened between V1 and V9 (88.54% to 92%) - there is no support for the view that evolutionary the 80:20 rule is strengthened.

Package	Version	FIN
database	V1	88.54
	V2	90.33
	V3	91.18
	V4	91.15
	V5	91.23
	V6	92.44
	V7	92.19
	V8	92.11
	V9	92.00

TABLE 2: 80:20 metrics for the SmallSQL system

3.3. The DjVu System

Table 3 shows the data for the DjVu system. Again, FIN appears prominently in table and so too does the FOUT metric. However, the PACK metric features in V8 of the DjVu package. One plausible explanation for the dominance of FIN and FOUT and an implication of that dominance is that it may render the use of other coupling forms unnecessary.

Package	Version	PACK	FOUT	FIN
DjVu	V5		80.23	
	V6		80.23	
	V7		80.46	
	V8	80.68		
Toolbar	V1			83.81
	V2			81.90

TABLE 3: 80:20 metrics for the DjVu system

It is interesting that none of the values in Table 3 overlap. Inspection of the raw data revealed that generally, when FIN was high, FOUT was low (and vice-versa)

3.4. The pBeans System

Table 4 shows the trends for the pBeans system. The FIN metric does not feature in the 80:20 rule in the pBeans package. It does, however, feature in the first eight versions of the data package. This suggests that the FIN and other forms of coupling may have a complementary relationship. When there is high proportion of 80:20 FIN relationships, there are a low number of other 80:20 forms of coupling. Tables 1 and 2 support this theory and in Table 3 the six 80:20 relationships are non-overlapping (further supporting this theory). In a practical sense, and one explanation for this phenomenon, is that if there are a high number of classes with large FIN values then by the *law of averages* there will be fewer classes with large numbers of FOUTs since the coupling that these latter classes need is satisfied by the classes with large FIN (and which they use). In other words, FIN and FOUT follow a 'hub' principle which minimises outgoings by maximising incomings. This would also support the use of 'key' classes (i.e., classes that contain a large amount of functionality that many other classes use) as a means of minimising coupling. In other words, and counter-intuitively, large classes (or classes with large amounts of coupling) can have a beneficial effect (if we assume minimising FOUT is an aspiration of developers).

Package	Version	MPC	EXT	PACK	FOUT	FIN
pBeans	V1	90.98	85.73	85.00	88.32	
	V2	90.79	85.67	85.00	88.32	
	V3	91.89	86.56	81.05	87.29	
	V4	91.89	86.56	81.05	87.29	
	V5	92.23	87.10	81.05	87.29	
	V6	92.23	87.10	81.05	87.29	
	V7	94.02	89.23	81.82	90.60	
	V8	89.50	84.67	83.90	85.69	
	V9	89.50	84.67	83.90	85.69	
	V10	89.54	84.74	83.90	85.69	
Data	V1					82.86

	V2					87.50
	V3					92.89
	V4					92.89
	V5					92.89
	V6					94.78
	V7					99.53
	V8	89.64	85.08		83.76	81.14
	V9	88.86	84.38		80.22	
	V10	88.26	83.85			

TABLE 4: 80:20 metrics for the pBeans system

Figure 1 shows the pBeans metrics over the course of *all* 20 versions from Table 4. The values follow the same trends for much of all 20 versions (except for RFC and FIN). Low values in the figure reflect a more even spread of coupling for that metric. Figure 2 shows the same values for the SmallSQL system. Coupling types appear to have the same pattern in both figures; inspection of the raw data for the other two systems revealed a similar trend. In other words, coupling remains relatively static for all systems as they evolve; an 80:20 rule is not exacerbated as a system evolves.

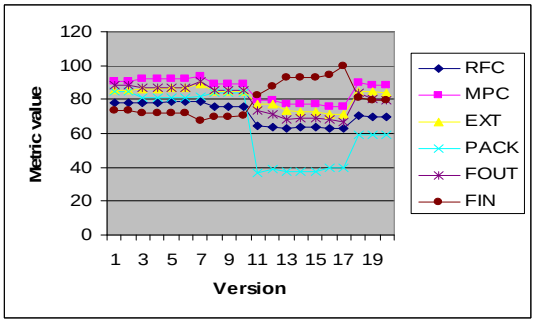

FIGURE 1: Metric values for pBeans

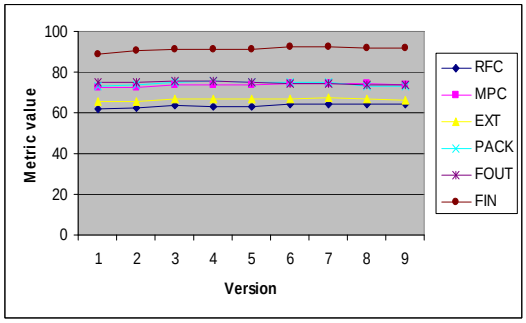

FIGURE 2: Metric values for SmallSQL

4. CONCLUSIONS

In this paper, we have explored the 80:20 in four open-source systems and six coupling metrics. The key findings were that certain metrics had a greater propensity for that rule than others, namely fan-in and, to a limited extent, fan-out. High use of these two features seemed to exclude the use of other types of coupling. Moreover, an 80:20 rule did not seem to worsen as a system evolved. Finally, we suggested that dominance of fan-in (particularly) might act as a 'hub' for 'key' classes and with which many other system classes communicate. We urge more studies into coupling and particularly evolutionary-based studies [7]; all the data upon which this study rests can be made available to other researchers upon request.

REFERENCES

[1] Bartsch, M. and Harrison, R. (2006) A coupling framework for AspectJ. *Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, Keele, UK, 10-11 April.
[2] Belady, L. and Lehman M. (1976) A model of large program development, *IBM Sys. Journal*, **15**(3), 225-252.
[3] Briand, L., et al. (1999) A unified framework for coupling measurement in OO systems. *IEEE Trans. on Soft. Eng.*, 25(1), 91-121.
[4] Briand, L., Devanbu, P. and Melo, W. (1997) An investigation into coupling measures for C++. *Proceedings of the 19th International Conference on Software Engineering (ICSE 97)*, Boston, USA, 17-23 May, pp. 412-421.
[5] Chidamber S. and Kemerer C. (1994) A metrics suite for OO design. *IEEE Trans. Soft. Eng.*, 20(6), 476–493.
[6] JHawk tool: (http://www.virtualmachinery.com/jhawkprod.htm).
[7] Kemerer, C.F. and Slaughter, S. (1999) Need for more longitudinal studies of software maintenance. *Empirical Software Engineering: An International Journal*, **2**(2), 109-118.
[8] Mubarak, A., Counsell, S., Hierons, R. and Hassoun, Y. (2007) Package evolvability and its relationship with refactoring. *Proceedings of Third International ERCIM Symposium on Software Evolution*, Paris, France.
[9] Wheeldon, R. and Counsell, S. (2003) Power law distributions in class relationships. *Proc. IEEE International Workshop on Source Code Analysis and Manipulation*, Amsterdam, The Netherlands, 26-27 September, pp. 45-54.