System Performance Analyses through Object-oriented Fault and Coupling Prisms

Alessandro Murgia*, Roberto Tonelli,
Michele Marchesi, Giulio Concas
Dept. of Electrical Engineering
University of Cagliari, Cagliari, Italy
*Dept. of Mathematics-Inf., University of Antwerp, Belgium
alessandro.murgia@uantwerpen.be

Steve Counsell, Stephen Swift
Dept. of Information Systems
Brunel University,
Uxbridge, Middlesex, UK
steve.counsell@brunel.ac.uk

ABSTRACT
A fundamental aspect of a system's performance over time is the number of faults it generates. The relationship between the software engineering concept of 'coupling' (i.e., the degree of inter-connectedness of a system's components) and faults is still a research question attracting attention and a relationship with strong implications for performance; excessive coupling is generally acknowledged to contribute to fault-proneness. In this paper, we explore the relationship between faults and coupling. Two releases from each of three open-source Eclipse projects (six releases in total) were used as an empirical basis and coupling and fault data extracted from those systems. A contrasting coupling profile between fault-free and fault-prone classes was observed and this result was statistically supported. Object-oriented (OO) classes with low values of fan-in (incoming coupling) and fan-out (outgoing coupling) appeared to support fault-free classes, while classes with high fan-out supported relatively fault-prone classes. We also considered size as an influence on fault-proneness. The study thus emphasizes the importance of minimizing coupling where possible (and particularly that of fan-out); failing to control coupling may store up problems for later in a system's life; equally, controlling class size should be a concomitant goal.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features.

General Terms
Measurement, Performance, Experimentation.

Keywords
Coupling, fan-in, fan-out, faults, refactoring.

ICPE'14, March 22–26, 2014, Dublin, Ireland.

http://dx.doi.org/10.1145/2568088.2568089

1. INTRODUCTION
As an object-oriented facet, excessive coupling [5] is, anecdotally and empirically, an acknowledged contributor to faults [1, 4] and hence a contributor to degradation of system performance. Here, coupling refers to the inter-connectedness of the components in a system. While no large system can exist without some form of coupling, wherever possible, developers should seek to minimize this facet of a system because of the relationship it forms with faults. In this paper, we explore the relationship between faults and coupling where the latter is decomposed into *fan-in* and *fan-out* [12]. We chose these two sub-forms of coupling to make the distinction between inward (fan-in) and outward (fan-out) links belonging to a class and we explore the relationship between fan-in and fan-out for sets of fault-free and fault-prone classes. Two releases from each of three Eclipse projects (six releases in total) were used as an empirical basis of the study and coupling and fault data extracted from those releases. Results showed a stark contrast between the coupling levels of fault-free classes and fault-prone classes. Low fan-in and fan-out appeared to support fault-free classes. Statistical evidence supported that result for four of the six releases studied. Analysis of class size between fault-prone and fault-free classes suggests that faults arise because of relatively high amounts of coupling (particularly fan-out), which in turn is dependent largely on class size. Developers should therefore aim to keep components as small as possible as a first line of defense against faults and pursue re-engineering and refactoring activities which decompose classes and methods.

1.1 Motivation
The motivation for the research comes firstly, from the relative lack of replication studies exploring the relationship between fault data and coupling on a longitudinal (i.e., release-by-release) basis [13]. Secondly, it comes from the fact that the relative merits of fan-in or fan-out (and combinations thereof) are still open research areas. Study of fan-in and fan-out from a fault perspective might lead to novel conclusions about the relationships between faults and other software engineering disciplines such as refactoring [2, 10, 18]. Finally, the work is also motivated by the role that class size plays in class composition. Lessons and conclusions based on class data are of questionable value unless size is taken into consideration [8, 9]. The indirect link between faults and class size (through the medium of coupling) is often neglected by other studies, yet can offer us insights into system behaviour otherwise lost [8]. The remainder of the paper is organised as follows. In the next section, we

describe the analysis of the data on a release-by-release basis supported with fault and coupling data. We further support that analysis with statistical correlation. In Section 3, we discuss issues raised by the study including threats to study validity and related work before concluding in Section 4 pointing to further work.

2. DATA ANALYSIS

2.1 Preliminaries

Our analysis is based on two releases from each of three Eclipse projects: jdt.core, jdt.ui and jdt.uiworkbench. We used Eclipse as a basis of our research since it is a large, long-surviving system with more than ten years of development. We took into account faults between six releases. The JHawk tool [14] was used to extract the incoming 'fan-in' coupling and outgoing 'fan-out' coupling metrics for each class. The RefFinder tool [16] was used to extract up to sixty-three refactorings between two releases and the data reported relates to all classes that had been the subject of at least one refactoring between releases. We chose to study classes which had been refactored as opposed to studying every class in the system for two reasons. Firstly, by studying refactored classes, we obtain a mix of those classes that are likely to have been problematic and those that have had refactoring applied to them in a perfective sense. Secondly, because the study presented is part of a wider examination of refactoring, faults and the relationship these two have with coupling [15]. Fault data was collected manually by one of the researchers and subsequently verified by another. In the subsequent analysis, we present tabular data between each release, relevant correlation values, level of significance (1% or 5%) and data for fan-in and fan-out to support relationships between coupling and faults. We define a fault in this paper as an 'observed failure in the system' and marked as such by Eclipse developers using the Bugzilla fault-tracking system. We use parametric correlation measures assuming a normal distribution (Pearson's) and non-parametric measures which make no assumption about the data distribution (Kendall's and Spearman's). Using all three gives a broad and complete set of correlation values that using one alone might not accord. Finally, we note that the set of faults and refactorings collected were disjoint across releases; in other words, double counting of either was not a threat to the collected data.

2.2 Jdt.core_3.0_3.1

Table 1 shows summary statistics for the values of fan-in and fan-out (henceforward called 'FIN' and 'FOUT') for all 1151 classes containing at least one fault and which had been refactored at least once between releases 3.0 and 3.1. It also shows the FIN and FOUT data for the 154 fault-free, refactored classes. For each set, the minimum (Min), maximum (Max), Mean and standard deviation (SD) values are shown. For example, for the set of fault-prone classes, the minimum FIN was 0, maximum FIN was 228 with mean FIN 20.21 and SD 29.91. For fault-prone classes, we see that the mean value of FOUT is greater than the mean for FIN.

Table 1. Coupling data for jdt.core 3.0_3.1

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	228	20.21	29.91
FOUT	0	195	30.59	32.69
Faults	1	71	10.44	32.12
Fault-free				
FIN	0	23	6.38	5.81
FOUT	0	34	7.63	6.57

The maximum value of FIN of 228 was for a class with 54 methods and 2 faults. The FOUT value of 195 belonged to a class with 159 methods and which exhibited 28 faults over the period studied. The maximum number of faults (71) belonged to a class called Scope with 81 methods. Its FIN was 66 and FOUT 48.

For the set of fault-free classes in the same table, the maximum values of FIN and FOUT are noticeably lower than those for fault-prone classes presented. The mean FIN value of 6.38 and mean FOUT value of 7.63 are considerably lower than the corresponding mean values for fault-prone classes. The question as to whether 'fault-free' classes over that period (as opposed to fault-prone classes) present a different profile in terms of their FIN and FOUT values also arises. To determine the relationship between FIN and FOUT in each of the categories, we correlated the respective values for the two data sets (fault-prone and fault-free). Correlation values between FIN and FOUT for fault-free classes showed no statistical significance for any of the three coefficients (0.03, 0.05 and 0.08 for Pearson's, Kendall's and Spearman's, respectively). On the other hand, for the set of fault-prone classes, we found correlations of 0.10 for Pearson's (not significant), 0.14 and 0.19 for Kendall's and Spearman's coefficients between FIN and FOUT, the latter both significant at the 1% level (0.01). For this project, FIN and FOUT profiles for fault-prone classes seem to differ significantly from that of fault-free classes. Both the FIN and FOUT values for fault-prone classes are higher. From Table 1, fault-prone classes tend to have a higher mean FOUT than the corresponding FIN value. This would make sense; a class with many incoming couplings (i.e., a high FIN value) is *depended upon* by many classes for the functionality that it offers. This means that it should be maintained very carefully because of the ripple effect of faults that changes to that class would cause to those dependent classes. The same is not true of classes with a high number of outgoing couplings (i.e., high FOUT). In that case, because the dependencies are outgoing, the ripple effects of any faults are likely to be less severe and careless maintenance would have less of an effect on the system.

While the tabular data and correlation results do suggest that fault-prone classes have a higher FIN and FOUT than fault-free classes and this is certainly a feature of the release studied, we cannot overlook the fact that size is also an important factor in the determination of coupling and, indirectly, the number of faults in a system. To this effect, we computed the median and mean number of methods (NOM) for each of fault-prone and fault-free classes. For fault-prone classes, the mean NOM was found to be 44.24 with a median value 23; for fault-free classes, the mean NOM was significantly lower at 10.24 and median 7. In other words, there was a wide variation in the size of the classes between those exhibiting faults and those that did not exhibit faults. Clearly, size is a factor in determining coupling and with that comes faults; restricting class size growth (and with it

coupling) may be the major weapon against high fault incidence in classes. We could go further; based on the evidence presented we could hypothesise that the balance of coupling should be in favour of a higher FIN rather than use of FOUT. Developers should thus avoid building classes with high FOUT values.

2.3 Jdt.core_3.1_3.2

Table 2 shows statistics for FIN and FOUT for the 929 fault-prone, refactored classes between releases 3.1 and 3.2. It also shows the FIN and FOUT for the 130 fault-free refactored classes. For fault-prone classes, the mean FOUT is again noticeably larger than the mean of FIN. The maximum value of FOUT is for the same class as the previous release; the FIN value of 194 belonged to class with 52 methods and 11 faults. The maximum number of faults (38) belonged to the same class as in the previous release Scope, which now contained 84 methods. The FIN for this class was 73 and the FOUT 49. As to whether fault-free classes exhibit a different pattern to fault-prone classes, correlating FIN and FOUT for fault-free classes gave a value of 0.05 (not significant) for Pearson's and yet 0.16 and 0.21 for Kendall's and Spearman's, respectively. Both of these values were significant at the 5% level only. This contrasts with 0.07 (significant at the 5% level), 0.20 and 0.29 both significant at the 1% level for Pearson's, Kendall's and Spearman's, respectively for fault-prone classes.

Table 2. Coupling data for jdt core 3.1_3.2

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	194	22.72	33.84
FOUT	0	195	28.94	32.69
Faults	1	38	7.06	7.19
Fault-free				
FIN	0	121	5.97	12.71
FOUT	0	33	5.49	7.00

Again, we note a strong difference in the FIN and FOUT relationship depending on whether a class is fault-free or fault-prone. The mean FOUT for fault-prone classes is again higher than that of its corresponding FIN value. When we consider the class size between these releases (given by NOM), we see a similar pattern as that in the previous section. For fault-prone classes, the mean NOM was 46.66 and median 23; for fault-free classes, the mean NOM was 9.2 and median NOM 7. Size is clearly a major factor in the FIN and FOUT values of Table 2; in particular, faults seem to thrive in highly-coupled classes, a feature of large classes. It is interesting to see (from Table 2) a similarity in the FIN and FOUT mean values for fault-free classes (5.97 and 5.49). As noted in the previous section, a balanced coupling profile (avoiding high FOUT values) appears to be a feature of fault-free classes and of fault-free classes here.

2.4 Jdt.ui_3.0_3.1

Table 3 shows summary data for the 1489 fault-prone and 555 fault-free classes between releases 3.0 and 3.1 of jdt.ui. For the former, the maximum value of FIN was 196 belonging to a class with 51 methods and 10 faults (FOUT for this class was 42).

Table 3. Coupling data for jdt.ui 3.0_3.1

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	620	8.16	32.29
FOUT	0	51	12.57	10.61
Faults	1	23	3.60	3.37
Fault-free				
FIN	0	287	3.26	13.55
FOUT	0	22	5.19	4.45

The maximum value of FOUT was 51 for a class with 23 methods and 4 faults between releases. The maximum number of faults (23) was for class MoveInnerToTopRefactoring; this class had a FIN of 6 and a FOUT of 39. For FIN and FOUT, for the fault-free set of classes (mean FIN 3.26 and mean FOUT 5.19) we notice a distinct difference in the magnitude of these values compared with those of fault-prone classes.

For the fault-free set of classes, correlations between FIN and FOUT were -0.05 for Pearson's (not significant), 0.10 and 0.14 for Kendall's and Spearman's, respectively (both significant at the 1% level). The correlations for FIN versus FOUT for fault-prone classes on the other hand were 0.06 (significant at the 5% level), 0.09 and 0.11 (both significant at the 1% level). For this release, there is thus a parallel between FIN and FOUT for fault-free and fault-prone classes, in contrast to previous releases. When we again consider class size (given by NOM) we found that for fault-prone classes, the mean NOM was 31.69 and median 21; for fault-free classes, the mean NOM was just 8.53 and median 7. Again, size is a major factor in the FIN and FOUT values and the result extrapolated from Table 3. While we accept that faults arise because of high coupling, it is perhaps allowing class size to grow which is a key contributor to high coupling.

2.5 Jdt.ui_3.1_3.2

Table 4 shows the summary data for the 1187 fault-prone and 390 fault-free classes between releases 3.1 and 3.2. The FIN value of 698 was for the same class as the previous release, with 47 methods and 6 faults between releases.

Table 4. Coupling data for jdt.ui_31_32

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	698	12.68	57.17
FOUT	0	70	13.73	12.54
Faults	1	18	3.30	2.69
Fault-free				
FIN	0	84	4.51	11.85
FOUT	0	47	6.40	6.21

The FOUT of 70 belonged to a class consisting of 31 methods with 7 faults over the releases. The class with 18 faults was called JavaEditor and had no methods. Its FIN was 100 and it's FOUT 43, well above the mean of 13.73. From a fault-free class perspective, the mean FIN for the 390 classes was 4.51 and its mean FOUT 6.40. Correlations between FIN and FOUT for those classes were 0.06 (not significant), 0.16 and 0.23, both significant at the 1% level (Pearson's, Kendall's and Spearman's, respectively). This compares with 0.13, 0.15 and 0.20 for fault-

prone classes - all significant at the 1% level. Between these two releases, the relationship between fault-free and fault-prone classes was comparable. Considering the class size (given by NOM), we could see this result from a different perspective. For fault-prone classes, the mean NOM was 23.96 and median 17; for fault-free classes, the mean NOM was just 11.82 and the median 8. Again, size would appear to be a major factor in the FIN and FOUT values and from the result from Table 4.

2.6 Jdt.uiworkbench_3.0_3.1

Table 5 shows the summary data for the 695 fault-prone and 154 fault-free classes between 3.0 and 3.1. For the set of fault-prone classes and, as *per* other releases, the mean FOUT of 16.24 exceeds the corresponding value for FIN (9.14). The class with a FIN of 196 was a class with 51 methods and 10 faults over the course of the releases; its FOUT was 42. The maximum value of FOUT was for a class with 99 methods and 24 faults; its FIN was 30. The class with the highest number of faults was a class called WorkbenchPage with 179 methods and 30 faults.

Table 5. Coupling data for jdt.uiworkbench_30_31

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	196	9.14	16.91
FOUT	0	83	16.24	18.88
Faults	1	30	4.79	5.91
Fault-free				
FIN	0	23	3.51	7.09
FOUT	0	76	5.34	4.30

The mean FIN for fault-free classes was 3.51 and that for FOUT 5.34. A difference between FIN and FOUT between fault-free and fault-prone classes is thus evident. The correlations for this set of classes were -0.10, -0.03 and -0.07 (Pearson's, Kendall's and Spearman's), none of which were significant. These values contrast starkly with the correlation values for the set of fault-prone classes between FIN and FOUT of 0.59, 0.27 and 0.36, all significant at the 1% level. Considering the class size (given by NOM), for the set of fault-prone classes, the mean NOM was 27.14 and median 16; for fault-free classes, the mean NOM was just 9.56 and median NOM 6. From the data presented, size is a major factor in the determination of FIN and FOUT and, by implication, faults.

2.7 Jdtui.workbench_3.1_3.2

Table 6 shows data between releases 3.1 and 3.2 for the set of 419 fault-prone and 124 fault-free classes. In keeping with the other releases, the mean FOUT for fault-prone classes (18.44) far exceeds that of FIN. The maximum FIN value was 204 and this belonged to same class as in the previous release. It exhibited 4 faults over the period studied. The maximum value of FOUT was 102 - the same class as the previous release with FIN value of 36; faults for this class fell to 19 over the period. The maximum number of faults was 24 for a class called WorkbenchWindow with 144 methods; its FIN was 44 and it's FOUT 89. For the set of fault-free classes, the mean FIN for fault-free classes was 3.68 and for FOUT 5.88. The correlations between FIN and FOUT were -0.07, 0.09 and 0.12, none of which were significant. Correlations between FIN and FOUT for fault-prone classes, on the other hand, were 0.56. 0.34 and 0.45 (for Pearson's, Kendall's and Spearman's, respectively), all significant at the 1% level. As per most of the releases, there is a clear distinction between the relationship between FIN and FOUT, depending on whether a class is fault-free or fault-prone.

Table 6. Coupling data for jdt.uiworkbench 3.1_3.2

Classes	Min	Max	Mean	SD
Fault-prone				
FIN	0	204	9.04	21.54
FOUT	0	102	18.44	23.23
Faults	1	24	4.47	4.97
Fault-free				
FIN	0	67	3.68	7.99
FOUT	0	28	5.88	6.31

When we once again consider the class size (given by NOM), we found the mean NOM to be 30.00 and median 18; for fault-free classes, the mean NOM was 12.07 and the median 7. Size is once again a major factor in the FIN and FOUT values and from the result from Table 6.

3. DISCUSSION

The study has highlighted the differences between FIN and FOUT for fault-free *vis-a-vis* fault-prone classes. Clearly, developers should try to avoid FOUT becoming excessively high. However, *one guaranteed way of minimizing FOUT is to ensure that a class does not grow in size such that it needs to be coupled to so many other classes. We would condone the use of re-engineering and refactoring techniques to decompose classes should it be felt that a class is growing out of hand.* Of course, we have to be pragmatic; developers only have limited time to devote to such activities. The alternative of faulty classes, however, maybe a worse one. One justification for using FIN and FOUT in this paper is that we could not have emphasized the differences between these two types had we chosen to use CBO metric of Chidamber and Kemerer [6], for example. Figure 1 captures the mean FIN values for fault-prone and fault-free classes across the six releases studied abstracted from Tables 1-6. The dashed line is the set of FIN values for fault-prone classes in the six releases studied; the un-dashed line is that for fault-free classes.

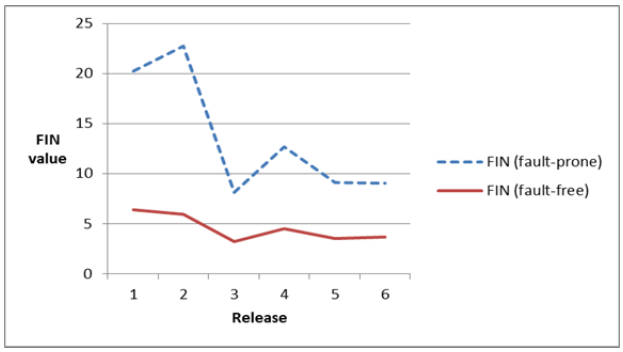

Figure 1. FIN values for fault-prone and fault-free classes (six releases)

There is a clear difference between the set of FIN for the two sets of classes and this applies to all releases. FOUT values for fault-prone classes clearly exceed those of fault-free classes. Figure 2 shows the FOUT mean values for fault-prone and fault-free classes abstracted from Tables 1-6.

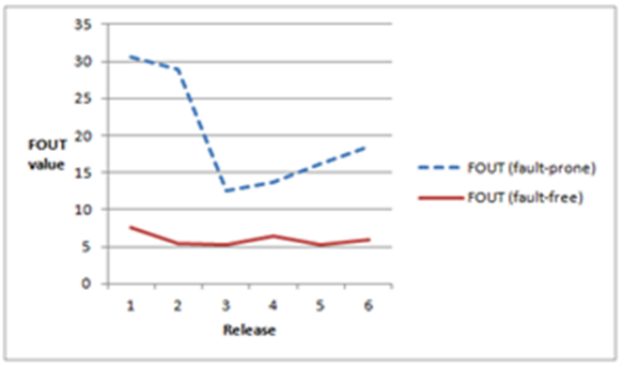

Figure 2. FOUT values for fault-prone and fault-free classes (six releases)

Again, the dashed line represents the set of fault-prone classes and the un-dashed line the set of fault-free classes. Once again the difference between the two sets of classes is clear. It is interesting to note from Figures 1 and 2 that the gap between mean values is most pronounced at earlier releases of the system. While this may be purely chance, it might suggest that re-engineering or refactoring activity [10] may have been put in place to narrow the gap between the two. The extract class and extract method refactorings are just two of the standard set of 72 refactorings proposed by Fowler [10] which may have been applicable here.

On the other hand, the mean values (after falling dramatically in release 3) then start to rise in both Figures 1 and 2. While the results from this data analysis support the view that low FIN and FOUT values contribute to fault-free classes, we need to consider the view that it is the size of a class which may determine a) the level of coupling in a class and b) the fault-proneness of a class. To emphasize the difference, Figure 3 shows just the mean NOM values for fault-prone and fault-free classes as described in Sections 2.2–2.7. There is a clear trend for the number of methods in fault-prone classes to be larger than those that are fault free.

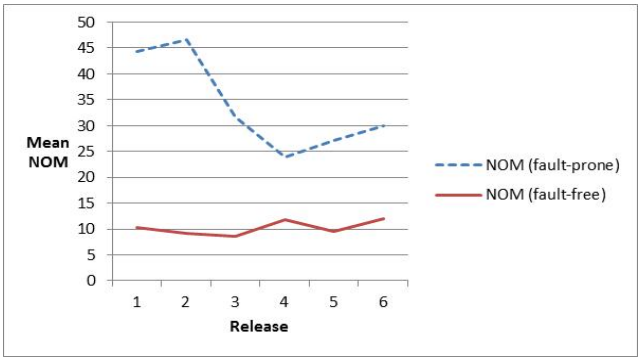

Figure 3. NOM for fault-prone and fault-free classes

3.1 Threats to validity

A number of threats to the validity of the research also have to be considered. Firstly, we only consider one system in our analysis and this could be considered a threat to external validity (i.e., the ability to draw conclusions based on a single system). However, an equal criticism could be applied had we used other systems in terms of our ability to compare results. Second, we have studied only limited releases of that system representing a small time frame in the overall life of the system. Third, in this paper we considered a class to be fault-free if it had not exhibited any faults *up until* that point in time. That is not to say, however, that the class will remain of that status – in a subsequent releases it may become faulty. For the purposes of the study however, this represents a useful benchmark against which we can judge fault-prone classes. Finally, we chose NOM as a mechanism for assessing size, but accept that there are other measures that might have been as appropriate as measures of class size. For example, lines of code (LOC); the problem with using LOC as a measure however is the wide variation with which it can be computed [17]. The NOM metric provides a standard measure of class size.

3.2 Related work

Faults or 'bugs' have been an effective measure of software performance since its inception [9]. The link between fault propensity and software complexity (in this case in the guise of coupling) is also well-understood. Study of fan-in and fan-out in this paper marks a departure from normal studies of the more generic 'coupling' form. In most coupling studies, the Coupling between Objects (CBO) metric of Chidamber and Kemerer [8] has been the standard coupling measure employed. However, while useful, the CBO metric does not distinguish between incoming and outgoing class coupling. This means that conclusions about the *direction* of coupling and the relationship with faults (in the case of the paper presented) cannot be made. Fan in and fan-out were initially presented by [12]. In this paper, we also consider class size as a factor in consideration of the results. This is based on the premises that as classes and methods grow in size, so too do coupling and therefore faults. The relationship between coupling and faults has been empirically shown to exist in OO software by Basili et al., and specifically in the C++ language by Briand et al., [4]; a framework for the measurement of coupling is provided in [3]. The work of El Emam on the influence of size as a confounding factor is also drawn upon herein; in the paper, it is proposed that size should be taken into account as a confounding variable when validating object-oriented metrics [1]. In this paper, we explored fan-in and fan-out but in the context of size also. The link between design patterns and fault-proneness was explored by Gatrell et al. [11]. Design pattern classes (i.e., those forming the pattern's essential structure) were found to be more fault-prone than classes that were not part of the pattern. Finally, much of the research on fault-proneness has focused on fault prediction [13]; the study presented adds weight to the view that size needs to be considered as an essential factor in any study of faults.

4. CONCLUSIONS/FUTURE WORK

The incidence of faults in a system over time is a reflection of the quality of the system and whether its performance continues to match its initial and ongoing requirements. In this paper, data between six releases of Eclipse was analyzed and fault and coupling data collected from each. Results showed that the fan-in

and fan-out pattern/profiles for fault-free *vis-a-vis* fault-prone classes showed notable and significant differences. Low values of fan-in/fan-out seem to promote fault-free classes. Equally, a high fan-out value seemed to predominate in fault-prone classes. One explanation for this feature of the data might be that, conceivably, it is easier to make additions and changes to a class which is not dependent on a large number of classes than one that is. The risk of amending a class with many incoming dependencies is significantly higher because of those dependencies. The message it would appear to a developer is to try, whenever possible to minimise coupling both from an incoming and outgoing perspective as would befit good software engineering practice. However, high values of fan-out should be especially avoided since from the evidence presented they are the classes which tend to evolve into fault-prone classes. It would also appear when scrutinising the data more carefully that the main factor in the high fan-in and fan-out classes is the size of a class. Developers should therefore seek to minimise the size of a class (through re-engineering or refactoring) thereby maintaining its cohesion as well as keeping coupling low. In other words, coupling and size co-evolve and *both* should be monitored.

In terms of future work, these emerging results need to be investigated through analysis of more systems and more releases. In this way, a body of evidence can be constructed. The relationship between specific refactorings and faults is one of several lines of research that this study informs as a partial replication to recent work [15]. We encourage further replication studies of this type as collaborative efforts as a means of generating a body of knowledge in this area. To that end, all the data used for the study presented can be made available upon request of the lead of the Brunel team member listed. Future work will explore other class characteristics for relationships with faults. In particular, whether a Zipf (or 80:20) Law exists between faults and other class features [19] such as cohesion [7] as well as fan-in and fan-out.

5. ACKNOWLEDGEMENTS

The research of Alessandro Murgia is sponsored by the Institute for the Promotion of Innovation through Science and Technology in Flanders through a project entitled Change-centric Quality Assurance (CHAQ) with number 120028.

6. REFERENCES

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

[2] Bavota, G., De Carluccio, B., De Lucia, A., Di Penta, M., Oliveto, R., Strollo, O., When does a Refactoring Induce Bugs? An Empirical Study Proceedings Working Conference on Source Code Analysis and Manipulation, Riva del Garda, Italy, 2012.

[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., Melo, W., An investigation into coupling measures for C++, in 19th International Conference on Software Engineering, Boston, USA, pp. 412-421, 1997.

[5] Briand, L., Daly, J. Wust, J., A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Soft. Engineering, 25: 91-121, 1999.

[6] Chidamber, S.R., Kemerer, C.F., A metrics suite for object oriented design, IEEE Trans. on Soft, Engineering, 20:476-493, 1994.

[7] Counsell, S., Swift, S., Crampton, J., The interpretation and utility of three cohesion metrics for object-oriented design. ACM Transactions on Softw. Eng. Methodol. 15(2): 123-149 2006.

[8] El Emam, K., Benlarbi, S., Goel, N., S. Rai: The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. IEEE Trans. Software Eng. 27(7): 630-650 (2001).

[9] Fenton, N., Pfleeger, S., Software Metrics: A Rigorous and Practical Approach', International Thomson Computer Press, 1996.

[10] Fowler, M., Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999.

[11] Gatrell, M., Counsell, S., Design patterns and fault-proneness: a study of commercial C# software, IEEE International Conference on Research Challenges in Information Science, Guadeloupe, 1-8, 2011

[12] Henry, S., Kafura, D., Software Structure Metrics Based on Information Flow, IEEE Transactions on Soft. Engineering 5:510 – 518, 1981.

[13] Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., A Systematic Literature Review on Fault Prediction Performance in Software Engineering IEEE Transactions on Software Engineering, 2012.

[14] JHawk tool: www.virtualmachinery.com/jhawkprod.htm). 2013.

[15] Murgia, A. Tonelli, R., Counsell, S., Concas, G., Marchesi, M., An Empirical Study of Refactoring in the Context of Fan-in and Fan-out: an Empirical Study. Proceedings of European Conference on Software Engineering, Szeged, Hungary, March 2012.

[16] Prete, K., Rachatasumrit, N., Sudan, N., Kim, M., Template-based Reconstruction of Complex Refactorings, International Conference on Software Maintenance, Timisoara, Romania, pp. 1-10, 2010.

[17] Rosenberg, J., Some Misconceptions About Lines of Code, IEEE International Software Metrics Symposium, (METRICS '97), pages 137-143, Bethesda, Maryland, USA, 1997.

[18] Weißgerber, P., Diehl, S., Are refactorings less error-prone than other changes? Proceedings of the International Workshop on Mining software repositories, pages 112–118. ACM, 2006.

[19] Zipf, G., Human Behavior and the Principle of Least Effort, Addison-Wesley, 1949.