Object-oriented cohesion as a surrogate of software comprehension: an empirical study

Steve Counsell, Stephen Swift, Allan Tucker,
Department of Information Systems and Computing, Brunel University,
Uxbridge, Middlesex. UB8 3PH.
Email: steve.counsell@brunel.ac.uk
Tel: +44 (0) 1895 266740

Emilia Mendes, *Department of Computer Science, University of Auckland, New Zealand*

Abstract

The concept of software cohesion in both the procedural and object-oriented paradigm is well known and documented. What is not so well known or documented is the perception of what empirically constitutes a cohesive 'unit' by software engineers. In this paper, we describe an empirical investigation using object-oriented (OO) classes as a basis. Twenty-four subjects (drawn from IT experienced and IT inexperienced groups) were asked to rate ten classes sampled from two industrial systems in terms of their overall cohesiveness; a class environment was used to carry out the study. Four key results were observed. Firstly, class size (when expressed in terms of number of methods) did not tend to influence the perception of cohesion by any subjects. Secondly, well-commented classes were rated most highly amongst both IT experienced and inexperienced subjects. Thirdly, the empirical study suggests that cohesion comprises a combination of various class factors including low coupling, small numbers of attributes and well-commented methods, rather than any single, individual class feature per se. Finally, the research supports the view that cohesion is a subjective concept reflecting a cognitive combination of class features; as such it is a surrogate for class comprehension.

1. Introduction

The concept of software cohesion has its roots in the 1970's when Stevens et al. [15] started looking at inter-module metrics for procedural software. Yourdon and Constantine later categorised cohesion on a seven point ordinal scale from functional at one end to coincidental at the other [17]. Since then, various attempts in the object-oriented community have been made to capture cohesion through software metrics [1, 8, 10]. The best known and most investigated of these metrics is the Lack of COhesion in Methods of a class (LCOM) proposed by Chidamber and Kemerer (C&K) [8]. The LCOM metric rates a class as cohesive if every method uses every instance variable; at the other extreme, a class whose methods use disjoint instance variables is considered uncohesive.

Despite these attempts at capturing class cohesion and valuable work contributing to our understanding of cohesion [4], one gap in our knowledge persists, and that is an understanding of how software engineers view and rate cohesion on an empirical basis. In this paper, we empirically investigate class cohesiveness, using twenty-four subjects as a basis in a controlled classroom environment. Three hypotheses were investigated; the first related to the influence class size had on perceived cohesion. The second hypothesis related to the role of developer comment lines embedded in the classes studied. A final hypothesis assessed the influence of IT experience on the rating of cohesion.

Some interesting and counter-intuitive results were found as a result of the study, in particular reinforcement of the view that, empirically, class cohesion is effectively a combination of various class features. The subjective nature of cohesion implies that there is unlikely to be any generally accepted

Proceedings of the 2005 Fifth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'05)
0-7695-2292-0/05 $20.00 © 2005 IEEE

definition of OO class cohesion. Rather, a set of broad guidelines for attaining 'cohesive' classes. Furthermore, the subjective nature of cohesion suggests that it is a surrogate for comprehension – since many of the features identified by the subjects as contributing to cohesive classes in this study are obvious candidates for describing the comprehensiveness of a class. The results of our study also reinforce the view that observations from previous empirical studies are a useful guide for ongoing empirical studies.

The paper is arranged as follows. In Section 2, we describe the motivation for our study and review some related work in the area. In Section 3, we describe our hypotheses and describe the study itself. We provide analysis of the data in Section 4 and a discussion of the issues raised by the study in Section 5. Finally, we draw some conclusions and point to some future work (Section 6).

2. Motivation and related work

The motivation for the work described in this study stems from a number of sources. Firstly, the related concept of software coupling is relatively easy to both quantify and assess, whether in the procedural or object-oriented paradigm [6, 5]. Yet, a common understanding of what factors make a class cohesive has not been achieved by the OO software metrics community. Furthermore, to our knowledge, no empirical studies of the type described in this paper have been undertaken so far. We feel our study redresses this deficiency a little.

A second motivation for our study is to inform our understanding of how developers (both with and without experience) view software characteristics. If, as a community, we want to build more reliable and maintainable software, then we need to understand how, generally speaking, developers think and behave. In this paper, we adopt the stance that there is no obvious metric that encompasses all views of what constitutes cohesion. Our study however, may shed some light on, or reinforce developer guidelines and good practice for producing robust, easily-understood OO classes. In addition, through the use of appropriate metrics, we hope to inform our understanding of software quality issues [7].

A final motivation stems from previous work by the authors [10], where a measurement of cohesion based on Hamming Distance was found to correlate strongly with an association-based coupling metric. In other words, we hypothesised that cohesion and coupling were strongly inter-related in the OO paradigm. The work in the paper herein tries to uncover the extent to which this is true from a developer's viewpoint of cohesion. We are also aware of the criticisms that using only student subjects as a basis of an empirical study have received. In this paper, the majority of subjects had significant experience of industrial software development (enrolled on an advanced Master's programme), limiting to some extent this criticism. The study compares and contrasts the rating of cohesion by experienced and inexperienced IT subjects.

In terms of other related work, a number of attempts have been made to capture cohesion through software metrics. As well as the C&K LCOM metric, the Cohesion Amongst the Methods of a Class metric (CAMC) of Bansiya et al. [1] was found to correlate with both LCOM *and* the views of three developers on what constituted cohesion. Bieman and Ott [3] demonstrated the measurement of functional cohesion in C software. Finally, Briand et al. [4], propose a framework for measurement of OO cohesion and conclude that many of the cohesion metrics proposed are in most cases not validated theoretically and even fewer validated empirically.

3. Study Design

3.1 Subjects used

The subjects of the empirical study were all Master's Degree students on an eleven week course covering analysis and design of large-scale information systems. Topics covered in this module included an analysis of the LCOM metric and other C&K metrics, a variation of the CAMC metric and a discussion of other techniques such as the Goal Question Metric approach of Basili et al. [2]. The role of coupling of different forms was also covered and discussed. The course material prior to this study was delivered on a five lecture basis over a total of fifteen hours. We would expect each subject to have a good understanding of cohesion, coupling and metric areas when the study was started.

Sixteen of the twenty-four subjects used had industrial IT experience of development work and in one case of those sixteen, just managerial IT experience. Every subject possessed a degree in Computer Science (as a pre-requisite for entry to the Master's course). Table 1 shows some summary data for the sixteen subjects with commercial development experience.

Statistic:	Min.	Max.	Med.	Mean
Experience:	11 months	25 years	7.25 years	7.08 years

Table 1: Summary of experience of subjects

3.2 Hypotheses investigated

The study conducted had three key hypotheses (H1 – H3). All three hypotheses were developed prior to the study, based on the intuition and experience of the authors. From a cohesion viewpoint:

1. H1: Smaller classes are more cohesive than larger classes. The measure of size used here is the number of methods (this measure includes private, public and protected methods as well as constructors and destructors). The hypothesis is based on the belief that firstly, if a class is small, then it contains only the methods it needs to carry out its tasks (i.e., it is not an amalgamation of different functionality). Secondly, if it is small, the class is unlikely to have evolved very much (on the assumption that classes grow in size over time). As such, it could be viewed as a well-constructed class.
2. H2: Classes with relatively large numbers of comment lines are more cohesive than those without (or fewer) comment lines. This hypothesis is based on the belief that comments help developers understand the code and contribute to ease of assessment and maintenance of that class. Commenting is generally considered good practice. As such, a class with comments is more likely to be written in accordance with sound practice (e.g., there is minimal coupling between the class and other classes; the methods of the class are also strongly related in some way).
3. H3: There is a difference between the ratings of cohesion made by subjects with IT experience and those without IT experience. This hypothesis is based on the view that if a class is 'poorly written', then the experienced subjects are more likely to adjudge that class as cohesive than inexperienced subjects. Equally, if a class is 'well-written', then it will be considered cohesive by experienced subjects while inexperienced subjects will be less likely to identify subtle features contributing to class cohesion.

3.3 Materials used and procedures

The twenty-four subjects were each given a set of the ten C++ class header files being analysed. Due to space considerations in this paper, the full ten classes are not contained herein (the full set is available at: www.dcs.bbk.uk/~steve/classes.htm). The ten classes were chosen at random from two industrial-sized C++ systems. The only restriction placed on the choice of these classes was that there had to be a relatively wide range of class size (in terms of number of methods and attributes), but at the same time not too wide a range as to bias the results of the study. The two systems were:

1. Rocket. A compiler consisting of 32.4 thousand lines of code and comprising 322 classes [16].
2. ET++. A user interface framework, consisting of approximately 56.3 thousand non-comment source lines and comprising 508 classes.

Seven of the classes were taken from the Rocket system and the remaining three from ET++. Those three classes were `Arc`, `ArcList` and `DDGNodePtrList`. The two systems themselves were chosen on the basis that, firstly, they represented two contrasting application domains. Secondly, a number of previous empirical studies have used the same systems [9, 10, 13]; the results from these other studies helped to inform our understanding of the results in this study.

Each set of ten classes given to a subject was randomly shuffled before being distributed to minimise bias due to fatigue or learning effects. The subjects were given approximately fifteen minutes to rate and mark for each class on a scale of 1 - 10 how cohesive they thought that class was (where 1 represents a minimally cohesive class and 10 a maximally cohesive class). Subjects were also asked, where they thought it appropriate and interesting, to comment on why they had given the cohesion value they had. The scripts were collected in after the fifteen minutes had elapsed.

4. Data analysis

4.1 Hypothesis H1

Hypothesis H1 investigated whether small classes (expressed in terms of number of methods) were more cohesive than smaller classes). To investigate Hypothesis H1, the median and average cohesion scores for each class were calculated and then ranked. Table 2 shows the ascending ranked position of the ten classes according to experienced subjects, the name of the class, the median cohesion value according to all twenty-four subjects' rating of the class, the experienced subjects average score awarded (Exp.), that of the inexperienced subjects (Inexp.) and the Number of Methods in that Class (NMC). For example, class `ApplnDialog` was rated least cohesive and class `DDGNodePtrList` rated the most cohesive of classes by experienced subjects.

Table 2 also contains a Number of ASsociations (NAS) metric defined as the number of unique classes to which the class under consideration is coupled. This metric includes coupling due to inheritance and through any other form of coupling, i.e., through aggregation, the return type of a method or the parameter of a method. The NAS metric also includes coupling due to the C++ friend facility, which features in one of the classes studied (i.e., `BagItem`). The NAS also includes self-reference coupling. An example of the latter would be where a return type or parameter of a method is of the same class as that in which it is defined. As an example in the `Alert` class of Figure 1, '`Alert`' itself is a parameter to the `MetaDef` method. The following class definition of `Alert` (a class used as part of the study) shows an NMC value of eight and an NAS value of six (i.e., coupling due to `Dialog`, `VObject`, `Alert`, `Bitmap`, `AlertType` and `Menu` classes).

```
Class Alert: public Dialog {
                VObject*text,
                *image, *buttons;
      public:
        Metadef(Alert);

        Alert(AlertType,byte  *text=
        0,Bitmap *bm= 0, …);
        ~Alert();
        Vobject *DoCreateDialog();
        int Show(char *fmt, …);
        int  ShowV(char*fmt,  va_list
        ap);
        class Menu *getMenu();
        void   InspectorId(char*buf,
        int sz);
};
```

Figure 1: The Alert class of the Rocket system.

After each NAS value in Table 2 is a bracketed value representing the NAS with self-references omitted. Table 2 also contains a Coupling Between Objects metric (CBO) of Chidamber and Kemerer [8] which differs from the NAS metric in one key respect: it counts all couplings to other classes without the uniqueness restriction. As such, the CBO is susceptible to multiple counts of the same coupling (which could be considered a criticism of the metric).

From the class `Alert` we would obtain a value of seven for the CBO metric and six for the NAS value, the difference due to the class `Vobject` being referred to twice in the class. Table 2 thus illustrates the important difference between the NAS and CBO metrics. Moreover, the CBO values would, alone, indicate that `Assoc` and `ArcList` were highly coupled. Yet they are not in reality. The majority of the coupling is shared between three classes, in each case one coupling of which is a self-coupling.

Position (Exp.)	Class Name	Median (both)	Avg. (Exp.)	Avg. (In exp.)	NMC	NAS	CBO
1.	`ApplnDialog`	2	3.55	3.86	5	5 (4)	6
2.	`Alert`	3.5	3.75	4.00	8	6 (5)	7
3.	`Dialog`	3	3.91	3.38	16	7 (6)	8
4.	`CycleItem`	4	4.33	5.00	15	10 (9)	18
5.	`Arc`	4.5	4.40	3.00	5	2 (2)	6
6.	`Bitmap`	5	4.82	4.75	23	7 (6)	12
7.	`BagItem`	5	4.95	5.29	12	8 (7)	14
8.	`Assoc`	5	5.13	4.25	12	4 (3)	18
9.	`ArcList`	6	5.35	3.29	9	3 (2)	10
10.	`DDGNodePtrList`	6	5.86	4.00	9	4 (3)	10

Table 2: The ten classes, their cohesion ratings and coupling features

Table 2 shows that for both groups of subjects, the size of the class given by the NMC values does not seem to influence the cohesion values produced. For experienced subjects, classes `ApplnDialog` and `Alert` have low cohesion ratings and are two of the smallest classes (5 and 8 methods, respectively). Equally, classes `Bitmap`, `BagItem` and `Assoc` have relatively high cohesion values for both groups (with 23, 12 and 12 methods, respectively). The median values generally follow the pattern of the experienced group in terms of ascending order.

Interestingly, for experienced subjects, coupling in terms of NAS values seems to have influenced their cohesion ratings: low coupling gives rise to high cohesion values. This does not appear to be true for the inexperienced group, where no clear pattern emerges. Classes which were rated highly by experienced subjects are notable for their high CBO values, suggesting that subjects do not consider multiple references to the same classes when considering cohesion.

In conclusion, we would not find support for Hypothesis H1. It is not true from the study described that small classes are more cohesive than larger ones. On the other hand, classes with low coupling (according to the NAS values) do seem to exhibit higher cohesion values (amongst experienced IT subjects); in addition, for the same classes, the CBO is relatively high. In other words, it would seem that classes with relatively low NAS coupling are rated highly even though they have a high CBO value (i.e., the coupling is shared among a few classes). For inexperienced subjects, no pattern emerges for hypothesis H1. Key to a high rating of cohesion for experienced subjects is thus a low amount of distinct coupled classes i.e., low NAS and high CBO values are not necessarily detrimental to the rating of cohesion of a class.

4.2 Hypothesis H2

Hypothesis H2 investigated whether classes with relatively large numbers of comment lines were more cohesive than those with fewer comment lines. The role that comment lines play in aiding a developer or maintainer is still an open research issue. Rosenberg [14] has cast doubt on the appropriateness and viability of lines of code in general as a metric; his doubts would readily extend to comment lines. Previous work by some of the authors has been done to eliminate comment lines around constructors as a result of code bloat [11]. The work was done as part of Kerievsky's 'refactoring of constructors to factory methods' [12], but elimination of the comment lines was only as a by-product of eliminating the constructors themselves and not intended directly [13]. Fowler [11] describes the role of a comment to describe why code is where it is, not what that code actually does.

In terms of the definition of a comment line herein, we make no distinction in terms of how the lines are distributed throughout the methods of the class (i.e., whether beginning, end or dispersed throughout). We consider a comment line as simply any non-executable line apart from a blank line. If a line wraps-around, we consider it as one comment line only.

Table 3 shows the number of comment lines (NCL) found in each of the ten C++ classes in the order of ascending cohesion value according to the IT experienced subjects. It also shows the position of the class in the rankings by inexperienced subjects (fourth column). For example, class `ApplnDialog` was rated least cohesive by the experienced subjects, had zero comment lines and yet was rated seventh by inexperienced subjects. This may highlight the difference in the way that the experienced subjects view cohesion (in contrast to the opinion of inexperienced subjects). Inexperienced subjects may rely more on comments as an aid to comprehension (although the result for class `Arc` seems to contradict that theory). Table 3 shows that classes with relatively larger numbers of comment lines (`ArcList` and `DDGNodePrList`) were generally considered by the experienced subjects to be cohesive. The same is true of the inexperienced group. Clearly, the top two classes in terms of comment lines were ranked relatively highly in terms of their cohesion values by both groups. This would seem to indicate that comment lines are an aid to the assessment of cohesion (and comprehension) for both types of subject. However, in saying this, an allied factor (or even the critical factor in appraisal of cohesion) may be the low NAS values and high CBO value combinations for these classes (as discussed in Hypothesis H1). Such a low NAS may have given the subjects the impression of high cohesion. In other words, low coupling combined with a relatively large number of comment lines may together contribute to high class cohesion.

The fact that the methods of these two classes all contribute to a functional goal, i.e., the construction of a data structure, may also be significant in explaining their cohesion values. Nonetheless, we tentatively conclude in support of Hypothesis H2 that classes with relatively large numbers of comment lines are generally deemed more cohesive than those with fewer comment lines.

Position (Exp.)	Class Name	NCL	Position (Inexp,)
1.	`ApplnDialog`	0	7
2.	`Alert`	0	5
3.	`Dialog`	3	8
4.	`CycleItem`	0	2
5.	`Arc`	28	10
6.	`Bitmap`	0	3
7.	`BagItem`	3	1
8.	`Assoc`	3	4
9.	`ArcList`	47	9
10.	`DDGNodePtrList`	54	5

Table 3: Comment lines and associated cohesion positions

4.3 Hypothesis H3

Hypothesis H3 investigated whether there was a significant difference in their view of cohesion (by experienced and inexperienced subjects). From Table 2, we have seen that the two groups have differing views on cohesion. Hypothesis H3 determines whether across the two groups there is a consensus on what constitutes a cohesive class. To investigate Hypothesis 3, the values for each class within each of the two groups (experienced and inexperienced) were analysed. Table 4 shows the median values for those groups. The most revealing difference between the view of experienced subjects and inexperienced subjects occurs for the classes ranked 8, 9 and 10. Overall, only on two occasions is the median value of the inexperienced subjects greater than that of the experienced subjects (this occurs for classes `CycleItem` and `BagItem`).

Results from Table 4 therefore indicate that, generally speaking, experienced subjects tend to be more generous (and perhaps more forthright) in their assessment of cohesion. We thus find support for H3 and claim that there are substantial differences between the way that the two groups rate cohesion. This effect is particularly pronounced for the last two classes of Table 4.

Position	Class Name	Median (Exper.)	Median (Inexp.)
1.	`ApplnDialog`	3.5	1
2.	`Alert`	3	3
3.	`Dialog`	4	3
4.	`CycleItem`	4	5
5.	`Arc`	5	2
6.	`Bitmap`	5.5	5
7.	`BagItem`	4	5
8.	`Assoc`	5	3
9.	`ArcList`	7	3
10.	`DDGNodePtrList`	8	3

Table 4: Median values of classes for both groups of subjects

Examination of some of the comments provided by the experienced subjects on the classes reveals the motivation behind their assessment. For `ArcList`, comments such as 'has tight scope' (this subject rated cohesion value 9) and 'many methods seem to return the same variable' (rated cohesion value 8) were found. Equally, 'many dependencies' for `Alert` (cohesion rated 3) and 'has friends' for class `BagItem` (cohesion rated 1) pointed to some of the reasons for the subject giving low cohesion values for that class. We also note that comments about why a class was considered cohesive or otherwise tended to be made by the experienced subjects. On a final note, it is interesting that the classes `ArcList` and `DDGNodePtrList` both have zero instance variables. It would thus seem that minimising this feature is one pre-requisite for cohesively viewed classes. This contradicts the commonly-held view that instance variables are key to definition of class cohesion metrics.

5. Discussion

5.1 Threats to validity

A number of issues arise as a result of this study. The threats to its validity need to be considered. Firstly, only ten C++ classes were considered in this study and there were an uneven number of subjects in the two groups. In defence of these threats, we claim that it is very difficult to get any industrial developers to spend time on studies of this type. Realistically, it is rare to have the benefit of even modest numbers of each subject type. Secondly, only the header files were given to the subjects (not the full method definitions). In defence of this threat, we would claim that assessment of cohesion at the earlier design level is far more useful than after the class has been written at implementation level. A third threat to the validity of the study might have been the relatively short time available to the subjects (i.e., fifteen minutes) to complete their assessment of cohesion. In our defence, we believe that there are key indicators of a cohesive and uncohesive class which can be spotted quite quickly from paper versions of those classes. We feel that fifteen minutes was adequate and that is supported by full responses from most subjects. Finally, C++ was chosen because it is a core industrial OO language. We do accept that Java or C# would have been equally applicable and valid languages to use in this study.

5.2 Results

The results in this paper effectively reinforce what we already know or suspect about OO cohesion. It is an elusive and subjective concept. It is interesting that the features of classes considered cohesive by the subjects herein had very few instance variables (in the top two cases, there were zero variables). The maximum number of variables in any one class was three (for classes `Alert` and `CycleItem`); this typified the classes in the Rocket and ET++ systems. It would appear that subjects considered features other than instance variable usage when considering cohesion (although certain annotated comments by subjects did allude to this feature). The study raises a large number of issues on how subjects view cohesion. In one sense, we could easily replace the word 'cohesion' in this paper with the word 'comprehension'. We feel that one is a surrogate of the other. However, more studies on this topic and the other issues raised need to be undertaken before any concrete conclusions can be drawn.

Drawing on our knowledge of the two systems studied, previous studies have found ET++ to conform far more rigidly to sound OO practice than Rocket in terms of the way its classes are designed [9, 13]. It thus comes as no surprise that the classes in ET++ system (`Arc`, `ArcList` and `DDGNodePtrList`) fared so well. This also implies that if a system is exhibiting features which would be considered poor programming practice, then assessment of cohesion in that system is likely to follow the same trend. Finally, we do refer to the terms 'inexperienced' and 'experienced' subjects throughout the paper. We feel that both groups have a huge amount to offer in terms of their interpretation of cohesion. The word 'inexperienced' is not meant in any negative sense. In the next section, we draw some conclusions and point to future work.

6. Conclusions and Future work

In this paper, we have described a study which attempted to clarify the contributing factors to a cohesive OO class. Twenty-four subjects of mixed experience were used as a basis. Results from three hypotheses suggest that size, when measured in terms of number of methods per class is not a contributing factor in subjects' view of cohesion. Secondly, comment lines whether independently, subconsciously or as a contributing feature cause subjects to rate classes as having high cohesion. Finally, differences were found between the IT experienced and inexperienced groups in terms of the rating of class cohesion. When taken together, classes with low coupling, relatively higher numbers of comment lines and methods which contribute to a common goal of the class in a 'functional' sense seem to be indicative of a cohesive class.

In terms of future work, we hope to replicate this study at a later date; inclusion of the standard cohesion types as proposed by Yourdon and Constantine [17] would be an interesting slant on this work. We also need to consider in more detail the role that comment lines really do play in aiding the developer. Finally, the work in this paper is an ongoing project to assess the value and characteristics of IT experience and secondly, the traits of subjects without any IT experience.

Acknowledgements

We gratefully acknowledge the help of Jim Bieman at Colorado State University for access to the two systems investigated (ET++ and Rocket).

References

[1] J. Bansiya, L. Etzkorn, C. Davis and W. Li. A class cohesion metric for object-oriented designs. Journal of Object-Oriented Programming (January), pages 47-52, 1999.

[2] V.R Basili, G. Caldiera and H.D Rombach, The Goal Question Metric Approach. Encyclopedia of Software Engineering, Volume 1, pages 528-532, 1996.

[3] J. M. Bieman and L. Ott. Measuring functional cohesion. IEEE Transactions on Software Engineering, 20(8): 644-657,1994.

[4] L. Briand, J. Daly and J. Wust. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering Journal, 3(1): 65-117, 1998.

[5] L. Briand, J. Daly and J. Wust. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering. Volume 25(1): 91-121, 1999.

[6] L. Briand, P. Devanbu and W. Melo. An investigation into coupling measures for C++. In Proceedings of the 19th International Conference on Software Engineering (ICSE 97), Boston, USA. Pages 412-421, 1997.

[7] L. Briand, J. Wust, J. Daly and V. Porter, Exploring the relationships between design measures and software quality in object-oriented systems. The Journal of Systems and Software 2000, volume 51, pages 245-273.

[8] S. R. Chidamber and C.F. Kemerer. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6): 467-493, 1994.

[9] S. Counsell, G. Loizou, R. Najjar and K. Mannock. On the relationship between encapsulation, inheritance and friends in C++ software. Proceedings of the International Conference on Software Systems Engineering and its Applications, Paris, France, 2003.

[10] S. Counsell, E. Mendes and S. Swift, Comprehension of Object-oriented Software Cohesion: the empirical quagmire, Proceedings of the 10th International Workshop on Program Comprehension (IWPC 2002). Paris, France, pages 33-42, 2002.

[11] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, Reading, Massachusetts, 1999.

[12] J. Kerievsky, Refactoring to Patterns, Industrial Logic, online at: www.industriallogic.com, 2002.

[13] R. Najjar, S. Counsell, G. Loizou and K. Mannock, The role of constructors in the context of refactoring object-oriented systems, Proceedings of the 7th European Conference on Software Maintenance and Reengineering, 2003, Benevento, Italy, pages 111-120.

[14] J. Rosenberg, Some misconceptions about lines of code, Proceedings of the Fourth IEEE International Software Metrics Symposium, Albuquerque, New Mexico, pages 137-142, 1997.

[15] W. P. Stevens, G. J. Myers and L. L Constantine. Structured Design. IBM Systems Journal, 13(2): 115-139, 1974.

[16] A. Weinand, E. Gamma and R. Marty. ET++ - an object-oriented application framework in C++,. Proceedings of Object-oriented Programming Systems, Languages and Applications (OOPSLA), San Diego, USA, pages 46-57, 1988.

[17] E. Yourdon and L. Constantine, Structured Design, Prentice Hall, 1979.