Mechanisms for Interpretation of OO Systems Design Metrics

Philippe Li-Thiao-Té, Jessie Kennedy and John Owens
Department of Computer Studies, Napier University, Canal Court,
42 Craiglockhart Avenue, Edinburgh EH14 1LT, Scotland, UK
e-mail: {p.li, j.kennedy, j.owens}@dcs.napier.ac.uk
http://www.dcs.napier.ac.uk/osg
Tel: +44 (0)131-455 5340 Fax: +44 (0)131-455 5394

Abstract

A major criticism of metrics for object-oriented systems design has been their lack in generating meaningful feedback information. This paper proposes a data interpretation method based on visualisation of results obtained from the method redefinition metric set. The method encompasses an *"alarmer"* technique for identifying suspected design problems occurring under certain conditions. We demonstrate how the method provides a mechanism for metric result interpretation, design problem identification and design decision deduction.

Keywords: object-oriented metrics, data interpretation, visualisation, metrics profile, Smalltalk class hierarchy.

1. Introduction

"Goodness" of an Object-Oriented (OO) design depends on its interpretation and can therefore be the subject of debate. In comparison with the difficulty of designing an OO model, assessing it is also troublesome [Avotins94, Binkley96, Bristol96, Chidamber91, Koskimies92, Li96, McKim93]. The increased interest in metrics for OO has been significant over the last five years [Abbot94, Chidamber94, Harrison96, Henderson96, Kemerer96, Lewis95, Li93, Li97, Lorenz94, Whitty96]. However, little research has been done on the interpretation and analysis of metric results, making meaningful design decisions difficult. Thresholds, averages and benchmarks constitute the main elements against which comparisons are possible [Chidamber94, Henderson96, Lorenz94] but there are no general standards. The data visualisation method and alarmer technique presented in this paper are based on the idea of *metric profiles* i.e. typical shapes obtained for a particular metric. Any deviation from this "norm" will suggest potential inconsistencies which correspond to specific design problems. We chose to assess class hierarchies as this is the main structural and behavioural organisation for reusability and flexibility. Also, it is noticeable that simple mechanisms such as property inheritance, can generate complex behavioural problems which happen more often than expected [Cook92, Li97, Taivalsaari96]. The different concepts presented here are based on earlier work on the *multiple descendant redefinition (MDR)* problem and the *redefinition metric set* described in [Li97]. Briefly, the MDR problem contends that the property

Published in TOOLS Asia '97 conference proceedings, Beijing , China, 22-25 Sept. 1997.

inheritance scheme is broken i.e. nothing is inherited from the ancestor classes, when methods are completely redefined multiple times in the same branch of the hierarchy. A simple redefinition metric set was proposed to pinpoint such inheritance anomalies. The results suggest that a typical redefinition profile might exist for different branches in the Smalltalk class hierarchy. The detection and localisation of potential behavioural design flaws was possible. This paper concerns the study of the interpretation of metric results.

In section 2, we will explore how the assessment of an OO design is possible. In section 3, a description of the multiple descendant redefinition problem is given. In section 4, many visualisations of the same metric results set are evaluated towards the identification of potential design problems. The technique of the alarmer is explained in section 5, followed by a presentation of our data interpretation system. Finally, we discuss related work on metric results interpretation for OO sytems and present further work.

2. Assessing OO characteristics

Stating that a design is good is only valid with respect to criteria such as coupling, cohesion, reuse, hierarchy structure, polymorphism and the property inheritance scheme [Chidamber94, Briand94, Li97, Lorenz94, Tegarden95]. However, few papers identify wrong use of OO design concepts where a metric has been provided to permit its detection [Barnes93, Brito95, Li97]. Usually, a suite of coherent metrics is recommended to be used within a measurement programme, however, dependencies between them are still unclear [Henderson96]. In this paper, the metrics used are aimed at understanding the inheritance mechanism and the effects produced in class hierarchies, correctness of which is generally considered from two viewpoints [Armstrong94]:

- structural: which concerns the common properties to be defined in a class,
- behavioural: which concerns the necessary functionality of a class.

Depending on the designer's modelling strategy, a compromise, sometimes difficult, has to be made. Metrics can assess this uncertainty providing that the characteristics to be measured are well-identified and defined [Kolewe93]. Current criticisms of OO metrics are that they only provide hints to the "goodness" of the design. We argue that a precise identification of suspected problems with valid metrics for its assessment will suggest obvious directions or solutions for design improvement. Therefore, metrics can be prescriptive.

The next section discusses the multiple descendant redefinition problem in order to illustrate our metric profiling technique.

3. The *multiple descendant redefinition (MDR)* problem and associated metrics

The construction of class hierarchies still remains a problem due to the constraints involved and the required criteria. Although the property inheritance scheme defined by the OO paradigm constitutes a powerful mechanism for achieving reusability and flexibility, it can also introduce inconsistent design which infringes the essence of inheritance [Cook92, Li97, Meyer88, Rumbaugh91, Rumbaugh96, Seidewitz96, Taivalsaari96]. Method redefinition is justifiable for realising polymorphism [Meyer88, Rumbaugh91], nevertheless, it seems unnatural, as the main goal of

inheritance is to inherit, not to redefine. Method redefinition is the simple syntactic mechanism by which it is possible for a subclass to re-implement partially or completely an inherited method from its ancestor classes. In [Li97], it was shown that a high rate of completely redefined methods occurs in the current Smalltalk class hierarchy which strongly suggested the problem of potential behavioural inheritance inconsistencies. Many methods defined in a parent class were found to be completely redefined many times down the hierarchy. This was referred to as the *"multiple descendant redefinition"* problem. In such cases, there are no benefits from inheritance, indicating that either the parent class contains poorly abstracted methods or that the subclasses are wrongly situated in the hierarchy thereby obliging the class to ignore inherited properties. A set of redefinition metrics was proposed in order to assess such anomalies. For this paper, only the three main metrics will be considered:

- PRM: percentage of redefined methods includes the methods completely redefined, extended and realised,
- PCRM: percentage of completely redefined methods (including the ones realised),
- PEM: percentage of extended methods.

Often, a deep analysis of the class hierarchy source code depicted that suspected methods can simply lack from code factorisation and thereby fall under the case of a complete redefinition instead of an expected extension. It was suggested that, due to the class dependencies problem and incremental software development, developers would prefer to re-write their own version.

4. Metric results visualisation and interpretation

To date, most research work on metrics has concentrated on the metrics themselves and does not exploit the results from different perspectives. The derivation of metrics tends to generate a large data set, as a result. Thus the motivation for this work is the problem of metric results interpretation. A graphical representation of raw data is the first natural step. Instead of a table of plain numbers which might be suitable in some cases, the main benefits of a visualisation is that it is easy to pinpoint disparities. We aim at evaluating different representations in order to identify the appropriateness of these with regard to the metric chosen and the design characteristics expected to be interpreted. The suitability of the visualisation type chosen determines the correctness of the interpretation. In addition, it is sometimes either convenient or necessary to transform raw data before being visualised. A pre-transformation of data is seen as a re-processing stage where the results from the transformation are expected to exhibit some desired features or peculiarities. The choice of the *transformer function* is outside the scope of this paper, however, transformations in the metrics domain are considered. An example of a typical transformation could be simple filtering functions permitting the distribution of the data in a particular way for representation. Providing that suitable visualisation of the metric results exists, it is possible to identify design inconsistencies manifested by disparities on the representation for the assessed characteristic. Thus, the notion of *pattern profiles*. The technique of *alarmers* are aimed at specifying and recognising such disparities. More generally, the identification of conditions

under which a disparity occurs is essential for design problem detection.

The variety of classical chart types offered by Microsoft Excel© covered our requirements for evaluating metrics result visualisation. Ideally, it is sought to recognise typical pattern profiles which would be classified for a particular metric and thereby, the corresponding design problems. Suggestions for design improvement would then be facilitated. A profile should exhibit some expected characteristics or properties related to the metric considered. Statistical methods are a possible solution to data interpretation e.g. variance, covariance, regression and correlation but these are not considered here [Henderson96]. An alternative choice is to look at the range of possible chart types available for evaluating their appropriateness against the concerned metric. Not all graphic representations are suitable for a given metric, the choice depends on its type, on its properties, on the characteristics to be measured and on the type of results expected. For example, the redefinition metric set measures the amount of redefined methods in a class hierarchy. The measure is taken level by level in the hierarchy and a percentage is returned for each level. Therefore, the type of results is discrete which prohibits the use of smooth curves. Instead, visual representations such as bar charts or scatter plots are the most suitable.

A case study assessing the inheritance of the Smalltalk class hierarchy is used to evaluate a range of visualisations to aid the interpretation of the redefinition metric set. The detection and localisation of the MDR problem was expected. The same data set i.e. redefinition metric results, will be used in order to keep elements of comparison consistent. The concerned Smalltalk branches are the Object and the GraphicObject branches. They were chosen for the reasons that they show completely different redefinition profiles and that potential design

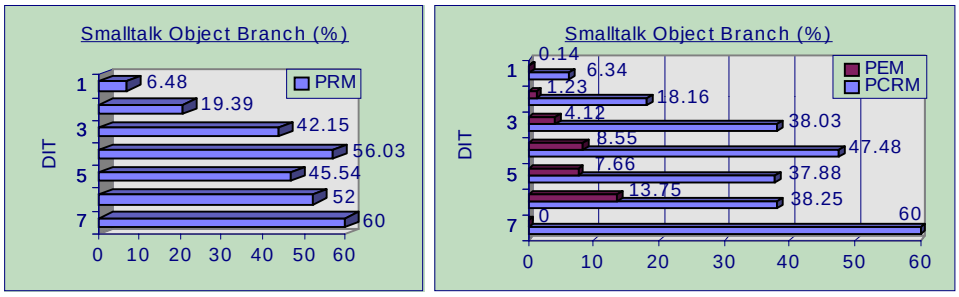

Fig. 1a and 1b: Bar chart profiles for the Object branch

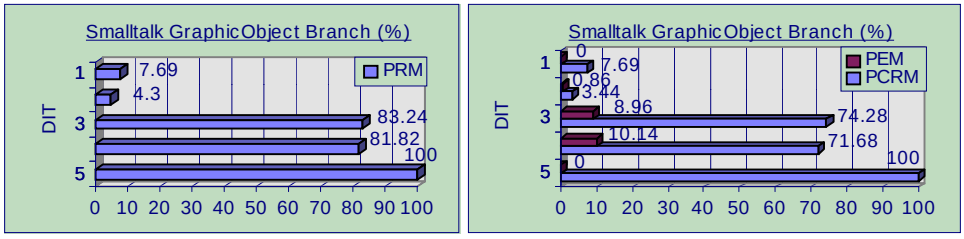

Fig. 2a and 2b: Bar chart profiles for the GraphicObject branch

problems exist in the latter.
In the following sections, a description of the main representations for the redefinition metrics are presented. The experiments were done on the Smalltalk Express[1] class library. The "OO system metric browser" described in [Li97] was used and extended in order to test the proposed alarmer technique.

4.1. Bar Charts

The PRM metric is the percentage of redefined methods without differentiation between its variants [Li97]. As expected, on Fig. 1a, the PRM rate of increase of the Object branch is fairly smooth. The first surprising feature (Fig. 1b) is the relatively high number of completely redefined methods (PCRM) in the whole Smalltalk hierarchy. Around 40% of methods are redefined from DIT=3 to DIT=7. The shape for the PEM is also smooth but the extension mechanism is not as highly used as expected with a mere 13.75% peak at DIT=6. Redefinition, which is recommended to be used with care, occurs frequently at all levels in the hierarchy, and extension, which is recommended, is rarely used. This raises the question of the correctness of the behavioural inheritance design. Also, the bar charts permit identification of suspect peaks occurring suddenly.
For the GraphicObject (Fig. 2b), the PCRM increases by a factor of 21.6 from DIT= 2 to DIT=3. This shows that method redefinition occurred at the top of the hierarchy, and questions whether the methods were initially well-abstracted. At this point, it is not possible to determine if the high level of redefinition is legitimate i.e. for realising polymorphic methods. A very suspect feature is depicted at DIT=5 on Fig. 2b: PCRM=100 and PEM=0 indicating that all classes at this level are completely redefining all their methods. Considering this level in the hierarchy, it is surprising that no methods were reused nor extended and that no addition of new methods were made. A deeper study of the GraphicObject branch reveals that this phenomenon seems to happen relatively often and concerns a few leaf classes. This abuse of the method redefinition mechanism seems to remedy the lack of multiple inheritance and seems to confirm our hypothesis.

4.2. Surface bar charts

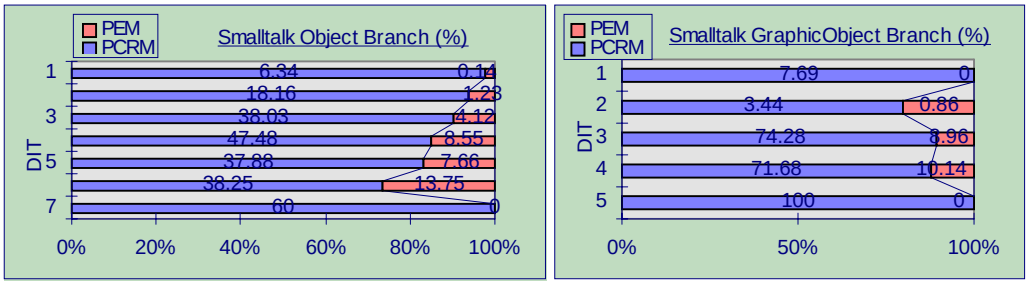

Fig. 3a and 3b: Surface bar profiles for the Object and GraphicObject branches

Surface bar charts are meant to exhibit the distribution of multiple metric result sets. On a bar chart scaled from 0 to 100%, it is possible to show the proportion of one or other metric for the system measured. This

[1] In this paper, Smalltalk Express™ designates the version based on Smalltalk/V® Win16 and WindowBuilder® Pro/V provided by ObjectShare®, a Division of ParcPlace, http://www.objectshare.com

representation is useful for metrics which are complementary, such as the PCRM and PEM metrics (both variants of the general definition of method redefinition). While the PCRM metric takes into account complete redefinition and realisation cases, the PEM includes the extension case. On Fig. 3b, the proportion of PCRM shown raises the issue of the balance between use of the mechanisms offered by OO technology. For the GraphicObject branch, the extension of methods is poor. This visualisation is convenient for depicting trade-offs in design where a compromise characteristic is expected to be seen. Notice that the join lines at the PCRM and the PEM boundary are drawn for ease of reading but do not define a smooth curve (the metrics results are discrete value sets). Further experiments on several other branches seem to confirm that the profiles shown occur on many occasions. An early analysis suggests two corresponding design problems:

- methods in top classes are poorly-abstracted,
- leaf classes are wrongly-subclassed as they are not reusing inherited properties.

A 100% of PCRM for the GraphicObject at DIT=5 suggests further analysis of the amount of polymorphic methods in the top classes. Comparing Fig. 3a and 1b the visual effect is immediate in the former. A quick glance permits the interpretation of the behaviour of distribution of PCRM and PEM down the hierarchy.

4.3. Surface charts

The surface charts are a variant of the surface bar charts but this representation is convenient for metrics returning non-discrete values. Although, not suitable for the redefinition metrics this is intended only as an example. Again, this visualisation can quickly outline the balance between two or more correlated metrics. Design decisions can therefore, be quickly deduced. On Fig. 4a, at DIT=6, the apportionment of PCRM vs. PEM is 60 to 40 % whereas at DIT=7, the apportion comes to respectively 100 to 0%. This suggests that leaf classes are more subject to complete redefinition than extension but no conclusions can be drawn on the correctness of the behaviour.

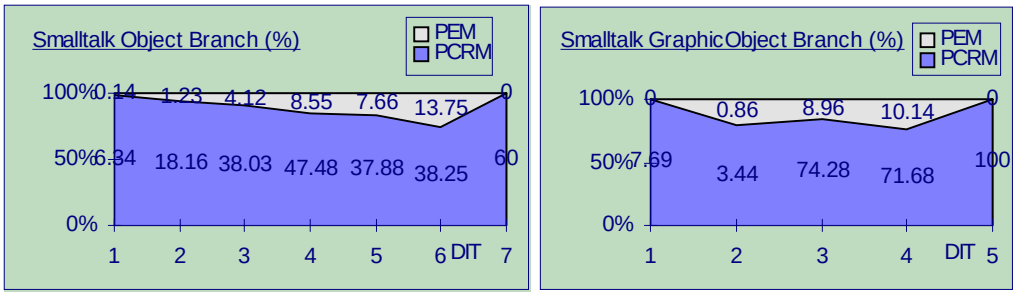

Fig. 4a and 4b: Surface profiles for the Object and GraphicObject branches

4.4. Addition bar charts

The addition bar charts are also suitable for complementary or related metrics. As completely redefined and extended methods are redefined methods, the sum of PCRM and PEM gives the PRM (Fig. 1a and 1b). On Fig. 5a and 5b, PRM is shown by the total extent of the bar. The addition bar chart is

considered an enhanced version of the simple bar chart as it makes clear the values for each of the shown metrics. However, to find out the apportionment for each of the metric, the surface charts are needed.

large amounts of information which can be confusing. For the GraphicObject branch, both curves obtained are rather intriguing as the redefinition activity seems to take place only in deeper levels of the hierarchy.

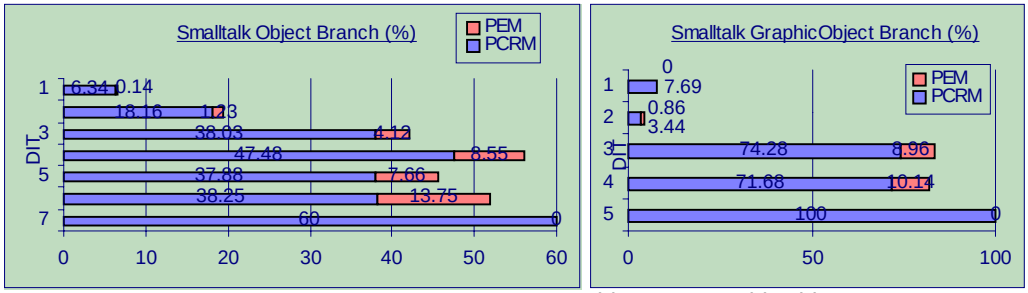

Fig. 5a and 5b: Addition bar charts profiles for the Object and GraphicObject branches

4.5. Radar charts

Finally, the radar charts are useful for exhibiting disparities between profiles. They can also pinpoint differences in the shape of the profile. In particular, it is convenient to use this representation when previous experiments have defined, for example, averages or thresholds for what is considered good or bad. Any disparity can then be quickly pinpointed. Again, the join lines are shown for ease of reading but it is possible to take them into consideration for identification of pattern profiles. When a smooth increasing curve is expected, the shape of the profile is a spiral. Attention should be taken when interpreting this type of chart as it can hold

Intuitively, this is confirmed by the assumption that a class situated deeper in a hierarchy inherits all methods from its ancestors classes. It is therefore potentially able to call a high number of possibly unrelated methods, thus explaining the high level of the redefinition. On Fig. 6b, it is clearly seen that two dimensions 1 and 2 are negligible compared to those remaining. Given that those dimensions represent the DIT level, it seems fair to conclude that a redefinition activity is more likely to happen in the bottom of the hierarchy due to the abstraction property of classes at the top. However, the rate of increase of the metrics cannot be easily pictured in those charts.

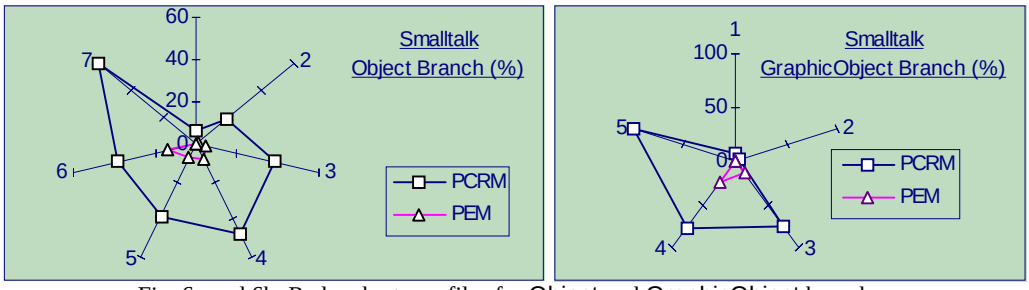

Fig. 6a and 6b: Radar charts profiles for Object and GraphicObject branches

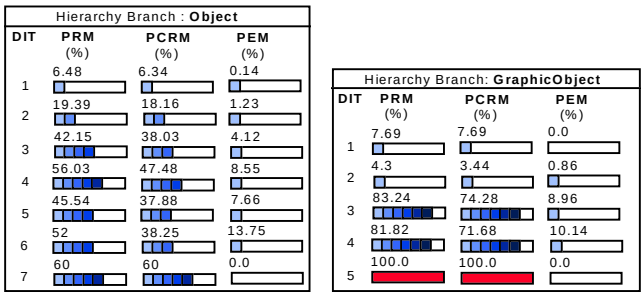

Fig. 7a and 7b: Colour coded bar for the Object and GraphicObject branches

Range	Apportion (%)	Colour coded bar representation
0	0	
1	0.01 - 14	
2	14.01 - 28	
3	28.01 - 42	
4	42.01 - 56	
5	56.01 - 70	
6	70.01 - 84	
7	84.01 - 100	

Table 1: Example of equally distributed ranges

4.6. A colour coded range bar charts

In the colour coded range charts, the metrics results set is pre-processed by a filter function first. The coloured bars shown in Fig. 7a and 7b have been obtained by checking the pre-defined ranges in which each metric value is situated. The coloured range bars are defined in Table 1. The apportionment has been arbitrarily chosen to be equal but this is not necessary. It is important to underline that this filtering method is not meant to be compared to a subjective assessment metric although based on the same principle as scaling. It is the responsibility of the designer to define the ranges, therefore the filter function. Table 1 shows an apportionment into 7 ranges, roughly equal to 100/7. When the proportions are equal, the smaller the proportion is, the closer this visualisation will be to the equivalent in a bar chart representation. In our example, colour shaded rectangles have been used to give a gradual effect. Also, it might be interesting to consider non-equal apportionment of the ranges. In such cases, attention should be given to the grounds on which the proportions are attributed to prevent from subjective interpretation [Henderson96]. For example, adopting a non-equal range strategy for a metric m and, providing that previous statistical experiments deducted a threshold of 60%, only three ranges are necessary. The first range is for 0, the second from 0 to 0.6 and the third 0.61 to 1. The same principle of colour coded rectangles can be used to quickly locate defects, thus only three colours would be used in this example.

In the GraphicObject branch, from DIT=2 to DIT=3, the peak (already pinpointed with the bar chart) appears even more suspect as the PRM increases by a factor of 21.6 suggesting

potential design flaws at DIT=2. Although this visualisation seems similar to the bar charts, but less accurate, the main idea for such a visualisation is to use it in conjunction with a triggering function or alarmer.

5. The concept of *"alarmers"*

The concept of an alarmer is simple. Suppose we want to detect any factor increase > 2 between two consecutive levels in the hierarchy. Any values satisfying the condition is expected to be pinpointed automatically. This is exactly what the alarmer technique is intended for. If an alarmer is set on for the GraphicObject branch in Fig. 7b, only the values of PRM and PCRM at DIT=3 would be found. If it was decided to use the colour coded bar charts for visualisation, only the two bars at DIT=3 for PRM and PCRM are shown. Indeed, the visual effect of the colour coded bar representation is immediate and asks for further analysis. The alarmer has accomplished its task in pinpointing the disparate results.

5.1. The alarmer mechanism

The first desired functionality of an alarmer is that it should provide a means for defining the behaviour to be detected. A simple form of an alarmer would be to detect a particular expected value within a set. In such a case, a simple condition function would be sufficient to filter the initial results set. For instance, this would be useful for comparing metrics results to the traditional averages or threshold numbers. Suppose that after some statistical analysis of the redefinition metrics results for a project, a threshold of 40% of redefinition is arbitrarily defined above which the design is to be re-considered. Therefore the triggering condition is simply:

metricValue >= AVERAGE _THRESHOLD

However, in the case of an alarmer triggered when the "weighted methods per class (WMC)" metric [Chidamber94] is greater or equal to 5. The triggering condition becomes a function:

wmc(class) >= AVERAGE _THRESHOLD

From the two cited examples, we can see that the core element of an alarmer resides in its triggering condition. In the case of large data sets, complex conditions can be applied. In a general case, an alarmer makes use of the following main components (Fig. 8):

- a filter function: when not all metric values are of interest in the whole metric result set, a filter function can be used to reduce the amount of data processed.
- a transformer function: if the data has to be transformed before application of the triggering condition, a transformer function e.g. statistical functions can pre-processed the metric results set.
- a triggering condition: defines the condition under which the set of values to check are satisfied.

6. Data interpretation system

A data interpretation system has been built based on the components shown on Fig. 8. The raw data in our model can be directly displayed or pre-processed before being displayed. The visualiser permits the display of the possible representations. A data transformer contains a list of functions permitting pre-processing of the data set. Typical transformer functions are filtering and statistical functions. When the designer has recognised some design problems in the hierarchy, the alarmer engine allows one to define and set up the alarm. In some cases, it would be necessary to pre-process the data set before setting up an alarm for the new metrics

set. Thus, the alarmer engine can co-operate with the data transformer.

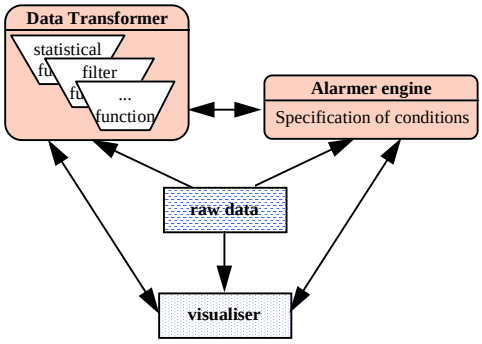

Fig. 8: Data interpretation system

7. Discussion and conclusion

Chidamber and Kemerer's [Chidamber91, Chidamber94] early work on OO metrics proposed a suite of six metrics for assessing the complexity of an OO model. Their metrics were applied on C++ and Smalltalk For each of their metric's suite, only simple histograms and summary statistics in a table form were produced. The interpretation of data relied on comparisons made between the histograms obtained for both sites. All charts represented the range of metric values (x-axis) obtained against the number of classes involved (y-axis) for each of the values. No correlation between the metrics were presented. The authors only suggest that a class hierarchy can be "top" or "bottom-heavy" i.e. the DIT metric and the "number of children (NOC)" metric are correlated. A high peak in the NOC histogram showed that most of the classes have no child classes. It was suggested that design practices dictated the use of shallow inheritance hierarchies, and that performance was the given reason in some cases. A use of surface bar charts might be a good candidate to exhibit previous observations. In such cases, it would be interesting to measure the number of classes per DIT level against their average number of children. Conceptually, it is expected to lead to the same conclusions.

In Lorenz and Kidd's [Lorenz94] project experience database, only histogram charts were used. In some cases, this type does not seem very appropriate due to the existence of large numbers in the results set. For instance, they considered the "number of message sends" metric and represented the values obtained against the number of methods. They correctly suggested that a rapid drop in numbers is the typical pattern found. This confirms the assumption that coupling between objects should be low in order to avoid inter-class dependencies. However, from a bad practise detection viewpoint, it would be more interesting to find out the methods which are strongly coupled. This could not be easily shown on the provided histogram as only a few methods are expected to send a large number of messages. Considering the colour coded range bar chart, an appropriate definition of ranges would immediately locate such disparate results for further analysis.

Currently, one of the main problems which infringes the development and adoption of OO metrics is a lack of tools for supporting their development and use in a general sense. We have developed a prototype which allows the localisation of suspected defect methods containing the MDR problem and the persistent storage metric results with a simple versioning mechanism.

This research work contributes an important part towards the improvement of data interpretation technique for OO metric results. Statistical analysis techniques provide solutions but are not easy to manipulate and not directed to OO design problems. Due to

the large data set which metrics can generate, it is necessary to have some mechanisms to quickly filter or re-process the data set in order to facilitate their interpretation. Detecting if a metric satisfies what it was aimed at can be blurred by the results if not interpreted correctly. In particular, this paper presented a description of many graphical representations for tackling the problem of interpretation. Visualisation techniques constitutes a powerful additional analysis tool for evaluating OO models. In addition, the alarmer technique provides an easy way to detect likely problems for a metric in detecting suspect values from a set of data when a triggering condition is satisfied. The use of the redefinition metric set as a basis for our case study demonstrated that the knowledge of redefinition profiles of a OO class hierarchy gave us insights into this important behavioural aspect. Precise detection of anomalies has been possible. Further tests are needed to explore the possibility of defining good or bad pattern profiles for these metrics. This new area of research seems to be promising and has to be considered as a whole part of the software measurement process as well as the software development life cycle. We believe that visualisation techniques and the concept of alarmers for data interpretation are strong and simple candidates for detecting complex design problems.

Future work encompasses:

- the creation of new types of possible representations,
- the classification of typical pattern profiles,
- the formalism of specification of the triggering condition for the alarmer, and,
- the integration of our proposed data interpretation system in a measurement framework model such as "the application of metrics to industry (AMI)" program proposed in [Rowe93].

The overall aim of the project is to propose an integrated development environment whereby measurement techniques are used to assess an OO model at early stage of the development and also to be able to re-inject design decisions into the model. Thus, the design-evaluation cycle can be completed and repeated. In summary, *"measure to understand, interpret to decide and transform to improve"*.

Acknowledgement: We would like to thank Mike Jackson (Wolverhampton University) for his comments on early versions of this paper.

References

[Abbot94] D H. Abbot, T D. Korson and J D. McGregor. *A Proposed Design Complexity Metric for Object-Oriented Development.* Department of Computer Science, Clemson University, Clemson, SC29634-1906, 1994.

[Armstrong94] J M. Armstrong and R J. Mitchell. *Uses and abuses of inheritance.* Software Engineering Journal, Janu. 1994.

[Avotins94] J Avotins. *Defining and Designing a Quality OO Metrics Suite.* TOOLS USA '94 Conference proceedings, USA, 1994.

[Barnes93] G M Barnes and B R Swin. *Inheriting software metrics.* Journal of Object-Oriented Programming, pp. 27-34, Nov/Dec. 1993.

[Binkley96] A B. Binkley and S R. Schach. *Impediments to the Effective Use of Metrics Within the Object-Oriented Paradigm.* OOPSLA '96 Workshop on "OO Product Metrics", Workshop report, 1996.

[Briand94] L Briand, S Morasca and V R. Basili. *Goal-Driven Definition of Product Metrics Based on Properties.* University of Maryland Institute for Advanced Computer Studies, Technical Report CS-TR-3346, Sept. 1994

[Bristol96] S R. Bristol. *Tools for Object-Oriented Metrics Collection*, OOPSLA '96 Workshop on OO Product Metrics, 1996.

[Brito95] F Brito e Abreu, M Goulão and R Esteves. *Towards the Design Quality Evaluation of Object-Oriented Software Systems.* Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, Octo. 1995.

[Chidamber91] S R. Chidamber and C F. Kemerer. *Towards a Metric Suite for Object-Oriented Design.* OOPSLA'91 Conference proceedings, pp. 197-211, Octo. 1991.

[Chidamber94] S R. Chidamber and C F. Kemerer. *A Metric Suite for Object Oriented Design.* IEEE Transactions on Software Engineering, 20(6), June 1994.

[Cook92] W R. Cook. *Interfaces and Specifications for the Smalltalk-80 Collection Classes.* OOPSLA '92 Conference proceedings, Vancouver, Canada, Oct. 18-22, ACM SIGPLAN 27(10):1-15, 1992.

[Harrison96] R Harrison and R Nithi. *An Empirical Evaluation Of Object-Oriented Design Metrics.* OOPSLA '96 Workshop on "OO Product Metrics", 1996.

[Henderson96] B Henderson-Sellers. *Object-Oriented Metrics, Measures of Complexity.* Prentice Hall Object-Oriented Series, ISBN 0-13-239872-9, 1996.

[Kemerer96] C F. Kemerer. *Measuring the Unmeasurable.* Bournemouth Metrics Workshop 1996, Bournemouth University.

[Kolewe93] R Kolewe. *Metrics in Object-Oriented Design and Programming.* Software Development, 1:53-62, Octo. 1993.

[Koskimies92] K Koskimies and J Vihavainen. *The problem of Unexpected Subclasses.* Journal of Object-Oriented Programming, pp. 53-59, Octo. 1992.

[Lewis95] J A. Lewis. *Quantified Object-Oriented Development: Conflict and Resolution.* 4th Software Quality Conference, 1:220-229, University of Abertay, Dundee, July 1995.

[Li93] W Li and S Henry. *Object-Oriented Metrics Which Predict Maintainability.* Journal of Software Systems, 23(2):117-122, 1993.

[Li96] P Li-Thiao-Té. *Integrating Measurement Techniques in An Object-Oriented Design Process.* Technical Report, Napier University, Edinburgh, 1996.

[Li97] P Li-Thiao-Té, J Kennedy and J Owens. *Assessing Inheritance for the Multiple Descendant Redefinition Problem in OO Systems.* To appear in OOIS '97 Conference proceedings, 1997.

[Lorenz94] M Lorenz and J Kidd. *Object-Oriented Software Metrics.* Prentice Hall Object Oriented Series, 1994.

[McKim93] J C. McKim, Jr. and D A. Mondou. *Class Interface Design: Designing for Correctness.* Journal of Systems and Software, (23):85-94, 1993.

[Meyer88] B Meyer. *Object-oriented Software Construction.* Prentice Hall International, C.A.R. Hoare, Series Editor, ISBN 0-13-629049-3, 1988. http://www.eiffel.com

[Rowe93] A Rowe and R Whitty. *Ami: promoting a quantitative approach to software management.* Software Quality Journal, (2):291-296, 1993.

[Rumbaugh91] J Rumbaugh, M Blaha, W Premerlani, F Eddy, and W Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs (N.J.), 1991.

[Rumbaugh96] J Rumbaugh. *A Matter of Intent: How to Define Subclasses.* Journal of Object-Oriented Programming, (18):5-9, Sept. 1996.

[Seidewitz96] E Seidewitz. *Controlling Inheritance.* Journal of Object-Oriented Programming, pp. 36-42, Janu. 1996.

[Taivalsaari96] A Taivalsaari. *On the Notion of Inheritance.* ACM Computing Surveys, 28(3):439-479, Sept. 1996.

[Tegarden95] D P. Tegarden, S D. Sheetz and D E. Monarchi. *A Sofware Complexity Model of Object-Oriented Systems.* Decision Support Systems: The International Journal, (13):241-262, 1995.

[Whitty96] R Whitty. *Object-Oriented Metrics: An Annoted Bibliography.* SIGPLAN Notices, http://www.sbu.ac.uk/~csse/publications/OOMetrics.html