Information for Authors

The International Journal of Applied Software Technology is a forum for the dissemination of pioneering research work in various areas of software technology. Both original work and tutorial exposition of permanent reference value are welcome. Papers are expected to be technical in nature, and will be reviewed by the advisory editors or competent independent referees.

Submission of a manuscript represents certification by the author that the article is not being considered nor has been published elsewhere. A manuscript largely based upon a conference paper must be so identified.

Format: Three copies of each manuscript should be submitted, including figures and tables. Manuscript should be typed double-spaced on A4 size paper with a minimum 1.25 in left margin. In general, articles should not exceed 6000 words. Articles should be subdivided into sections and sub-sections using short, meaningful subheads. A title sheet should precede the first page of the article and give the title of the article, name of authors(s), institutional mailing address (including department, institution, city, state, zip code, and country), or other affiliation. Include the telephone and fax numbers and e-mail address of the contact author. Number each page.

Disk Formats: If the manuscript was prepared on an IBM compatible personal computer, we prefer that a disk be sent with files in both the word processor used and the pure text (ASCII) file formats in addition to a hard copy. The word processor formats of Microsoft Word and Lotus AmiPro are preferred. Receipt of a file will decrease both the chance of error and publication time.

Abstract: An informative, comprehensive abstract of 75 to 200 words must accompany the manuscript. This abstract should succinctly summarize the major points of the paper, and the author's summary and/or conclusions.

Quotations: Quoted material of more than two lines should be set in a narrower width than the remainder of the text but continue to double space. At the close of the quotation, give the complete source including page numbers. Wherever necessary, it is the author's responsibility to obtain written permission from the copyright holder to quote or reproduce material from another publication.

Tables and Figures: All tables and figures must be submitted in camera-ready (publication quality) form. Tables and figures should be prepared on separate sheets at the end of the running text, i.e. Insert Figure 1 here. Numbers and figure captions at the top of each table and bottom of each figure. Photographs are welcome and should be black-and-white glossy prints.

Program Listings: Program listings should be avoided unless they are essential and add significantly to the article. Authors may wish to note in the article that programs are available from the author at a nominal cost.

Credits and Acknowledgements: Financial support for work reported or a grant under which a study was made should be noted. Any prior to the Acknowledgements. Acknowledgements or appreciation to individuals for assistance with the manuscript or with the material reported should be included. The acknowledgements are located at the end of the article.

References: Authors are responsible for checking the accuracy of all references and that all references cited in the text also appear in the Reference section. All references should be in alphabetical order by author (unnumbered), as shown below. Use the following style when referencing a book or an article in a periodical:


Submission: Send three copies of the manuscript to

Prof. Lu Si-Wt, (Editor-in-Chief)
Department of Computer Science
Memorial University of Newfoundland
St. John's, Newfoundland
Canada A1B 3X7

International Journal of Applied Software Technology
Volume 1, Number 3/4

Special Issue: Software Reuse

Contents

Structuring Requirement Specifications for Reuse
B Whitle, A Vickers, W Lam and J Mc Dermid 171

Reuse and Reusability Metrics in an Object Oriented Paradigm
G Succi, S Doublait, C Uhrik and F Baruchelli 191

Industrial Applications of the REBOOT Methodology
E Karlsson and G Sindre 203

The Development of a Software Clone Detector
N Dreyf, P Barson, S Field, R Frank and D Tansley 219

The Role of a Configuration Management System in a Reuse Oriented Framework
G Succi, C Uhrik, M Ronchetti, A Valerio and G Cardino 237

Causal Connectionist Support to Software Reusability
S Benlarbi and K Zeroual 249

Structuring Software Components based on Reusability
Y Park and D Ramjisingh 271

Design and Code Reuse Based on Fuzzy Classification of Components
E Damiani and M Fugini 291
Reuse and Reusability Metrics
in an Object Oriented Paradigm

G. Succi†, S. Doublait†, C. Uhrik†, F. Baruchelli†

† LII/DISy, Università di Trento
Via Zeni 8, I-38068 Rovereto, Italia
Fax: (+39) 464 443141
e-mail: {Giancarlo.Succi, Carli.Uhrik, Francesco.Baruchelli}@lidi.unika.it
+Sodalia S.p.A.
Via del Brennero 364, I-38100 Trento, Italia
Fax: (+39) 461 316663
e-mail: reuse@sodalia.it

Abstract: The connection between reuse and object oriented programming
(OOP) is outlined. Although both claim the same benefits, object oriented
programming can see “reuse” as a “test” demonstrating that the job of
OOP has been done. REUSE METRICS are seen as essential in measuring
the quality of object oriented designs and codes.

After a brief overview of reuse and metrics, the role of metrics in a reuse
based software development environment are described. Subsequently
attention is focused on specific figures of merit that can be used to judge the
quality of OOP designs and code for Reuse (actual reuse) and Reusability
(estimated reuse potential). Then a strategy based upon a flexible tool
platform is put forward for implementing the metrics. Finally, some conclud-
ing remarks are offered, and some other areas for additional use of
Reuse/Reusability metrics in management of OOP are suggested.

1. Introduction: Which Came First - the ReUSE Chicken or the OOP Egg?

Although the topics of object oriented paradigms and software reuse are closely
connected topics, when one begins to consider ways to implement and promote the
fullest advantages of reuse within an object oriented programming shop, the philo-
osophical stance to assume is not readily clear. In some sense, reuse is one of the
fundamental corner stones of object oriented programming (OOP) along with poly-
morphism, inheritance, encapsulation and abstract data types which effectively enable
it. Moreover, the benefits (perhaps over-zealously) attached to object oriented pro-
gramming are also the same benefits attributed to reuse:

International Jour. of Applied Software Technology Vol. 1 No. 3/4
• increased reliability - decrease in MTBF,
• increased usability - decrease in mean time between non-failure problem reports,
• increased maintainability - decrease in MTTR,
• increased understandability - decreased time/cost in all design-downstream life cycle stages,
• increased extensibility/flexibility - decreased mean time to make modifications or extensions,
• increased portability - decreased mean time for porting to new platforms.

In fact, all of these quality features can in some sense be considered as sub-components of REUSE as a quality figure, i.e.,

\[ \text{REUSE} = (\text{reliability, usability, maintainability, understandability, extensibility, portability}) \]

The relationship between OOP and REUSE derives from the fact that OOP strives to create largely decoupled or loosely coupled classes, and reusability is the "acid test" demonstrating that the job has been done right [Halliday 93]. Thus, one may view OOP is a means, whereas effective reuse is an end in itself. From this realization comes the natural suggestion that REUSE METRICS are essential in measuring the quality of object oriented designs and codes.

Despite the connection between reuse and OOP, reuse has a life of its own, not confined to OOP, so in considering whether reuse has been done well, one must not neglect the past experiences coming from outside the OOP sphere. Reuse in software engineering is a multifarious concept spanning a broad spectrum of aspects. It may center reuse in the large (reuse of systems components or middleware) or reuse in the small (sub-components). Assuming reuse in the small, one may adapt a generative approach (using shells or program generators) or more traditional programming. Assuming the traditional approach, one may emphasize design & development with reuse (i.e., reusing pre-existing components) or design & development for reuse (i.e., intending to design components for future reuse by others). This paper emphasizes both design & development "with reuse" and "for reuse" are of interest with attention to the 6 reuse qualities abovementioned (reliability, usability, maintainability, understandability, flexibility, flexibility) which can be associated as technically measurable aspects of an object oriented code.

2. The Role of Metrics in a Reuse Based Software Development Environment

Software metrics provide a means to quantify various aspects of software development. One can use these metrics in two ways:

• to control a software development process to ensure a consistent and stable process OR to control outputs of the various life cycle stages in the development of a specific product
• to estimate or predict the need for resources for a future scheduled product development OR to evaluate or assess a product/sub-product or process or individual or organization for its quality or performance in meeting specific objectives.

In this paper the emphasis is on product metrics which are essentially metrics for controlling quality. In the case of actual reuse, they are evaluative or assessment metrics, whereas in the case of reusability (or reuse potential), they are predictive.

In both cases, one may envision a project manager examining the results of the metrics against some threshold values and heuristic rules and subsequently deciding either to accept the software product or send it back with some comments to the author. Although it is easiest to imagine the decision regarding some acceptance of a code, there are metrics applicable also to designs and the other artifacts of the various stages of software life cycle - specifications, documentation, test plans, etc., which are also judgeable against metrics though these are presently less developed than those for code.

However in the case of an object oriented design and development, the earliest stages of the software life cycle (i.e., prior to coding) take the primary importance and should consume the lion's share of the resource investment if all is going right. This places an emphasis on design metrics over code metrics, particularly in a traditional waterfall model of software development. In the case of iterative development, design metrics are still important, but undesirable values for code metrics can serve to indicate faults to be overcome in the next generation or iteration of the system being developed.

There is another role for general metrics apart from reuse metrics which takes into account the past history of most large software development houses in a context of software reuse. Design-with-reuse in the absence of previously implemented design-for-reuse components implies that there must be accessible a significant body of so-called "legacy code" laying about in-house from past development efforts. However, typically such code is not of a quality satisfactory to the requirements of the new developments that would be candidates to reuse them, so they must be "re-engineered" to be adapted to a form compatible with reuse needs. In this situation, metrics are indispensable in highlighting problem areas that must be addressed in making an object oriented recasting of the previously existing legacy code without discarding their intrinsic value. For instance, metrics indicating complexity and coupling can help spot functions or subroutine components that need to be decomposed and can evaluate a proposed redesign for its merit. Metrics are the key to a good reengineering workbench, and where legacy code to OOP migration must occur to facilitate reuse, reengineering is inescapable.

It is a well known phenomenon that the non-reuse of software components is directly affected by the "not-invented-here syndrome". In the absence of any evidence to the contrary, a lack of confidence and assumed low quality unless otherwise known scores people out of reusable components [Poulin 94]. Giving suggestive evidence to the contrary is the form of quality/product metrics for the components under consideration should by-pass this effect. An example of this principle can be seen in the REBOOT qualifications such as "understandability" or "adaptability" which are quality features attached to items in the reuse repository in order to increase confidence in the items. Thus, reuse of components is promoteable through the introduction of such product metrics, provided there is some reliable technical basis for making the association to such metrics. This is true to some extent for reuse potential (reusability metrics) which are predictive, but it is true even to a greater extent for reuse metrics (i.e., reuse for items that have already actually been reused) because a highly reused item becomes even more reused the higher its reuse count gets (mainly due to a per-
3. Defining a Set of Metrics for Evaluating Reuse and Reusability

In general, there are two major classifications for metrics: "reuse proper" and "reusability". Reuse proper or "actual reuse" as a metric refers to an assessment or historical statistic which reflects in some way the number of times a piece of software has previously been used. The more a piece of software has been called or included with slight modifications (a relatively simple thing to do in an OOP context) in other programs, the higher any associated reuse metric will be.

On the other hand, reusability or "reuse potential" refers to a metric which is a prediction or estimate of the relative worth and applicability for reuse of an existing legacy code or a newly designed component regardless of whether it was intended for reuse by original design or as an after fact. This may either be based upon an estimate of the number of situations for reuse which should arise in a future time frame or upon relative qualities of merit that impact upon the ease with which a software component can be reused. The more chance there is that a piece of software could be reused in the future, the higher any associated reusability metric should be. The more that a piece of software is improved to enhance its quality from a reuse perspective likewise the more the associated reusability should be.

In specifically defining concrete instances of these two sorts of metrics in the OOP context, there are three approaches that seem rational:

1. start from the base concept of REUSE, in general, adapting previously defined reuse and reusability metrics to the OOP context,
2. start from the base concept of metrics in general, correlating predefined metrics with reuse and reusability to obtain heuristics about how indicative these metrics are for reuse & reusability,
3. start from the base concept of OOP, accepting the premise that good object oriented programming style automatically entails good reuse, many previously defined OOP metrics and associated heuristics should effectively be useful for reuse purposes.

In fact, a hybridization of the above sources of metrics is desirable, but for the point of clarity of the origins of any particular one, in the remaining discussion they are kept segregated.

A. Beginning from Reuse

Poulin gave a comprehensive tutorial on reuse/reusability metrics at the 1995 ACM-SIGSOFT sponsored symposium on reuse [Poulin 95]. A lot of the metrics are more for managerial purposes in cost estimation and success assessment, but some of the metrics have a technical basis. These metrics are summarized in Table 1 along with an assessment of their relevance to an object oriented paradigm. Rather than LOC (lines of code), it is advisable to use the equivalent of function points or some other complexity metric to size the code since particularly in OOP, lines of code are not very indicative as a complexity measure.

Many of the metrics in Table 1 also will be repeated in the following sections as well. Their purpose in being represented in multiple places is to highlight the point that the significance of these metrics for reuse in an OOP context arises from multiple perspectives and motivations, not from just a one dimensional line of justification.

B. Beginning from Metrics

Chen and Lee conducted an experiment in which they comparatively studied the traditional programming style against a reuse oriented style, in both cases using C++ as the programming language [Chen 93]. At the conclusion of the study, the benefits of reuse were measured and as well some widely recognized quality metrics were collected that characterise the style of the different groups involved. Though the purposes of the experiment were otherwise, this data effectively gives an empirical characterisation of reuse oriented desirable features reflected by conventional metrics. It suggests that it may be possible to use conventional quality metrics as predictors of reusability. Although this data is only suggestive and particular to C++, it warrants further examination. Table 2 summarizes the apparent connectivity between the metric quantities measured in the experiment and the capacity to discriminate between the conventional programming group and the reuse group (OOP reuse relevance is measured as average normalized deviation between the 2 populations).

Selby [Selby 99] also shows a similar correlation between easily measured conventional metrics and reusability according to NASA data showing statistical significance at a 95th following characterizations of reusable components:

- less than 140 source statements
- simple interfaces, few I/O parameters
- more calls to lower level functions, few calls to other modules
- less human interaction (user interface)
- good documentation in terms of comment to source statement ratio
- experienced few design changes during implementation
- more assignment statements than logic statements per LOC
- not necessarily low code complexity
- not necessarily depending on project size

all of which has some intuitive meaning as well.

C. Beginning from Object Orientedness

Lorenz and Kidd have implemented and described a number of extremely useful OOP metrics both for management purposes in estimating and in assessing the quality of designs and code [Lorenz 94]. From a more theoretical basis, Chidamber and Kemer introduce six metrics oriented towards OOP which have various desirable language independent properties [Chidamber 91]. In Table 3, is a brief synopsis of
<table>
<thead>
<tr>
<th>Type</th>
<th>Metric</th>
<th>Description/Comments</th>
<th>OOP* Rel.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REUSE METRICS</strong></td>
<td>Reuse Percent</td>
<td>(Reused LOC / Total LOC)*100 productivity with reuse / productivity without reuse</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>Reuse Leverage</td>
<td>t total objects used / new objects built (ave calls per object)</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>New Object Percent</td>
<td>(new objects built /total objects used in same app)*100</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>Reuse Level</td>
<td>(#reuse of internal item over threshold) + (#reuse of external item over threshold) / total items in system optionally items complexity w/ weightings</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>Reused Source Instructions (RSI)</td>
<td>Subtotal LOC</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Source Instructions Reused by Others (SIRBO)</td>
<td>Subtotal LOC</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>Reuse Value Added</td>
<td>(Total LOC + SIRBO)/( Total LOC - RS1)</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>Number of Inspections</td>
<td># of times someone considered the module for reuse</td>
<td>M</td>
</tr>
<tr>
<td><strong>REUSABILITY METRICS</strong></td>
<td>understandable</td>
<td>documentation + comments adequate</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>functionally complete</td>
<td>current + future expected requirements</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>reliable</td>
<td>implements advertised function w/o errors</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>errors/exceptions</td>
<td>isolates, documents, handles errors well</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>encapsulation</td>
<td>information hiding, details hidden</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>high cohesion</td>
<td>high specificity of functionality</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>low coupling</td>
<td>no external dependencies</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>portability</td>
<td>low dependency on system environment</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Empirica:</td>
<td>Program size should be small</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>McCabe cyclomatic complexity</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Flathead size</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

*Note: H=high, M=medium, L=low, H-L = highly inversely related, ...

Table 1: Proposed Reuse Metrics and their OOP Relevance.

<table>
<thead>
<tr>
<th>Type</th>
<th>Metric</th>
<th>Description/Comments</th>
<th>OOP* Rel.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Empirical</strong></td>
<td>Structure</td>
<td>entropy loading</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>control struct. entropy</td>
<td>links to other modules small (low coupling), low complexity</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>documentation</td>
<td>reuser's experience</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>generality</td>
<td>can multiply the value of other metrics</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>modularity</td>
<td>code/number of methods</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>environmental dependence</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>cyclomatic complexity</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>self descriptiveness</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>comments/ arc code checklist</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>documentation quality checklist</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>documentation /LOC checklist</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>module complexity</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>- far-in, far-out</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>- cyclomatic complexity</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>reliability</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>error tolerance</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td># problem reports</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Regularity</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Reuse frequency</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

*Note: H=high, M=medium, L=low, H-L = highly inversely related, ...

Table 1: Proposed Reuse Metrics and their OOP Relevance.
the Lorenz-Kidd design metrics and the Chidamber-Kemerer metrics rated by their relative significance for reuse purposes.

4. Implementing the Metrics

The process of implementing a metrication programme depends to a large extent upon a good understanding of what the goals are: in this case, promoting reuse. Once it has been agreed what this means, policies can be established to support the goal. Management must be aware of how the metrics can affect the reuse levels in project groups and be trained in interpreting the reuse metrics.

These activities - both at the training and the usage level - can in great part be facilitated by metrics tools, such as OOMetric [Lorenz 95] in the U.S. and the COSMOS Metrics Workbench [Fuchs 91] in Europe coming out of the ESPRIT COSMOS Project. The principle advantages of such graphically oriented tools are an open interface for flexibly defining and customizing freely chosen metrics through a script language and an ergonomically optimized design of the user interface which emphasizes a natural graphical format for efficient presentation of the metrics. This latter feature allows problem areas to be highlighted (using a number of heuristics that are built in) and thereby spotted quickly, subsequently proceeding to a selective deeper analysis (which effectively avoids wading through screens full of tedious numbers). OOMetric and COSMOS both represent suitably well-engineered software technologies for defining, collecting and analyzing metrics which can be used with OOP paradigms (e.g., C++ or SmallTalk). They have been successfully used by many large firms in the U.S. and in Europe such as IBM, ..., Alcatel, British Telecom, Dutch PTT, Cap Group, Philips, ING Bank and the Dutch Railway, and there is a growing body of experience with such tools.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description/Comments</th>
<th>OOP Reuse Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>productivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halstead size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>program volume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>program level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>estimated difficulty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>effort</td>
<td>McCabe</td>
<td></td>
</tr>
<tr>
<td>decision count</td>
<td>cyclometric complexity</td>
<td></td>
</tr>
<tr>
<td>reachability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dunsmore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>live variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable span</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: H=high, M=medium, L=low, H-1 = highly inversely related, ...

Table 2: Metrics & OOP Reuse Relevance Indicated in the Chen-Lee Experiment
5. Extensions and Future Areas for Research

As Poulin suggests [Poulin 90], a common model of reusability seems to be emerging. Regardless of the metrics to be used, the desired characteristics of a reusable item are the following: low complexity, well documented (i.e., high number of non-blank comment lines), few external dependencies (low fan-in and fan-out, few input and output parameters), proven reliability through testing and low error rates. Any metrics for reuse or reusability related to these characteristics are likely to be successful.

In considering the approach of measuring the correlation between reusability and more conventional quality metrics, there is an issue of whose programming style to measure in establishing the baseline reference for the estimator. Based on intuition gained from past work in using estimation metrics and noted diversity between object oriented languages [Lorenz 94], one can guess that the estimate is certainly language dependent as well as software house specific. Thus there will be no industry wide validity for any such measures, and heuristics or threshold will necessarily need be established locally. Thus only the methodology may be shareable.

With respect to reuse, a major issue is the sometimes floating evaluation of what is really to be considered reuse. At a first implementation of a piece of software, there may already be some "reuse" due to simply recalling a function or subroutine from many different contexts. This is referred to as "internal reuse" and generally has less significance than "external reuse" which amounts to cross-system or cross-application sharing of some software elements [Frakes 94]. Somewhere in-between in significance is the reuse of software elements across multiple generations of a system or application as it sometimes assumes more or less that of "internal reuse" and other times can be seen as "external reuse" because the context in which it is used is innovative enough to go beyond what it was used for in the first generation of the system/application. Here the point raised is that the present semiotics of reuse is a little fuzzy, so one wants to be careful not to take an overly syntactic definition and apply semantics filters to what one is counting as reusable (i.e., what constitutes a "reused line of source code") as well. The assessment of reuse and reusability should not be based on a single design perspective but should take into account many perspectives across multiple applications. In that way, it will be fulfilling its intended purpose - to provide relatively reliable figures of merit that will promote the production of good software at low cost.

There are additional non-technical or "project" aspects of reuse that are also beneficially quantifiable into metrics (and even OOP specific in many cases):

- measures that predict cost, time & resource estimation indicators,
- measures that can be easily connected to reuse incentive for rewarding individual performance,
- quantities that measure the presence of reuse inhibitor effects (for discovering problems),
- measures that indicate success of reuse at a corporate level,
- measures that rate the performance of reuse support organizations.

but these are much beyond the scope of the present paper.

In the end, it seems that reusability and reuse metrification are ill defined topics. It is largely context dependent, and for this reason one can not hope to establish an overall reusability metric. One can hope to empirically validate that various attributes of reuse or reusability have merit.

References


Feldkohn & McLuckey - SPA reference


