Issues in the Development and Application of Reuse Metrics in a Corporate Environment

Jeffrey S. Poulin

International Business Machines Corporation

Abstract

This paper describes the issues addressed when developing and implementing the IBM reuse metrics and return on investment model [25]. Recognizing the potential benefits of reuse has proven instrumental in inserting reuse in the IBM programming process. However, many issues arise in a large organization when implementing metrics due to the diversity in computer languages, environments, and cultures. This paper presents the options and methods to resolve many of these issues [29].

1.0 Overview

The uniqueness of reuse metrics comes from the fact that, unlike productivity metrics, they reflect on what a programmer *does not do.* Organizations reward those who deliver the most function, thereby encouraging developers to write code. In reuse we want to reward developers for *not* writing code.

This paper first discusses the goals and some criteria for reuse metrics. The paper then describes *what* to measure as reuse and *how* to measure reuse. This paper defines, from the business case perspective, what should count as a *Reused Source Instruction (RSI)* and then describes the issues that arise when putting the metrics into practice.

Organizational goals: Ultimately, an organization defines what to measure and report as reuse in accord with its goals [6]. Because IBM uses the metrics in a Return on Investment (ROI) model we believe they must reasonably quantify the level of reuse in an organization. Specifically, we have the following goals:

1. Reuse metrics must reflect effort saved.
2. Reuse metrics must encourage reuse.

The real difficulty in reuse metrics lies in judging whether a programmer *uses* or *reuses* software. Furthermore, as a technique becomes ingrained in a culture, it transitions in status from "novel" to "expected." What we reuse today we may simply "use" three years from now.

Criteria for reuse metrics: We must easily obtain, interpret, and implement the required data and metrics. The realities of the development process, organizational needs, and available data affect what we *can* report [10], [32]. Every company must consider the practical limits of their environment when constructing their metrics model.

2.0 What to Measure as Reuse

Use of base product (maintenance): Most software spends 80% of its life in maintenance [14]. Some might count maintenance as reuse because most of the software from a new release already existed [4]. However, when reporting maintenance metrics IBM excludes software from the product base[1] and only reports on new, changed, or deleted code. Reusable components completely new to the product contribute to the reuse level for the release, but calls to components in the base do not.

Use of Operating System (O.S.) services: Some count the use of O.S. services as a form of reuse because the O.S. raises the level of abstraction in programming [23]. However, we clearly do not expect to write a new O.S. for every new product [5], and therefore we do not count the use of an O.S. as reuse.

Use of tools: Programmers routinely use text editors, debuggers, compilers, and library systems. We use tools for their intended purpose; we do not consider the use of tools as reuse.

Use versus Reuse of components: Differentiating between "use" and "reuse" hinges on whether it saved the programmer from having to write code. Suppose a programmer retrieves a subroutine from a reuse library and uses it without modification. Although we count

[1] The product base consists of all code previously released in the product.

this as reuse, we do not count each subsequent call[2] to the subroutine as reuse.[3]

Use of pre-requisite products: Programmers have many utilities available to them, such as database systems and graphics packages. One *could* count the use of these utilities as reuse [13]. However, although these utilities save writing code, we do not count it as reuse [5]. We not only expect the use of these utilities, reporting use of a database system as reuse does not accurately reflect reuse activity.

Application Generators: Application Generators can create large quantities of code from specifications given in tables, pictures, templates, or fourth generation languages [9]. Some might consider the use of automatic program generators as a form of reuse [21]. Clearly, these tools provide an enormous advantage to the programmer.[4] However, we simply cannot compare this result with that of traditional methods; to obviate this problem IBM reports this code separately.

Use of High Level Languages (HLLs): HLLs could count as reuse because they raise the level of abstraction needed to program much as do abstract data types and class libraries [21]. However, we expect the use of HLLs and do not count it as reuse.

Code libraries: Counting reuse by source:

Use of utility libraries: The most basic libraries consist of the essential or required utilities in a given language (e.g., *stdio* in "C"). We expect their use and do not include them in RSI counts.

Use of standard libraries: Second come the local libraries every programmer must know to contribute to his group (e.g., memory functions in an O.S.). We consider these the local equivalent of utility libraries and do not generally count them as reuse.

Project libraries: Third, good design can result in well-structured programs and classes or superior use of language features such as generics and inheritance. Some "reuse" will result because the design or language encourage it. However, we have to recognize when we really save effort. We resolve the issue by deciding if the use falls outside the scope of normal good software design.

Domain-specific libraries: Fourth come the domain-specific libraries developed by organizations with extensive experience in an application area (e.g., flight control functions in aerospace) [31]. Although we expect a programmer in a domain to use these libraries, we count them as reuse. IBM encourages the development of domain libraries because they have proven to result in significant savings [22].

Corporate reuse libraries: Finally, reusable assets may reside in a shared, corporate database of fully tested, documented, and maintained components. We count the use of these parts as reuse.

White box versus Black box reuse: We find the benefits due to white box reuse (copying and modifying existing software) much less than those of black box reuse (use of unmodified parts). Although black box reuse costs more due to the additional effort required to make software reusable, we quickly recover this cost because we only have to maintain one part. At IBM we only count black box reuse.[5]

Porting versus Reuse: Formal reuse includes porting. However, counting porting as "reuse" may distort perceived benefits because we include porting in our business plans. A further reason to exclude porting comes from the fact that porting normally involves minor adaptations or simple recompilation. To include ported code in reuse metrics would result in very high reuse values; to prevent this distortion, we do not count porting as reuse.[6]

---

2 In one example, a group reported 512 calls to the same part from the same module as reuse.

3 Likewise, we may reuse class definitions in object-oriented languages but not the class instantiations.

4 One group reported a 95% reuse level on a 104 kloc product in which approximately 99 kloc came from a program generator.

5 Some organizations track the amount of white box reuse in their products to emphasize the "total leverage" gained by copying and modifying software.

6 Organizations tasked with porting software separately track the amount of porting they do in their projects.

Organizational boundaries: We expect programmers to use good software engineering principles and to share software within their groups. Normally, only the group which designed and maintains a component would use it in a new context or program. Other groups may never know about the component. Therefore, we seek to encourage reuse by counting a part as reused only when it crosses these organizational boundaries.[7]

Differences in organizations: In large organizations such as IBM there exists a diversity of expertise with regard to any technology, including reuse. The varying levels of reuse process maturity [18], [20] becomes an issue. Either an organization encourages formal reuse by adopting a very strict definition of reuse, or the programming culture has so fully accepted a model of reuse that they feel they must further adapt this reuse definition. Because we want everyone to adopt reuse, we focus less on achieving high levels of reuse than on constant improvement.

Table 1. Summary: What to measure as Reuse

Type of Reuse	**Measured?**
Maintenance	No
Operating System	No
Tools	No
Multiple Uses	One time
Prerequisite products	No
Application generators	Separately
High level languages	No
Utility libraries	No
Standard libraries	Maybe
Project libraries	Maybe
Domain-specific libraries	Yes
White box	No
Black box	Yes
Porting	Separately

3.0 How to Measure Reuse

Units of measurement: Numerous units in software exist: Lines of Code (LOC), Function Points (FPs) [2], [12], semantic tokens, equivalent lines of assembler, number of methods, number of features [24], number of classes, etc. However, most companies, including IBM, use LOC [7] despite their deficiencies [17]. Furthermore, LOC indicate overall code productivity and serve as a good secondary indicator in other phases [33]. Studies also show that methods such as FPs and LOC correlate highly [1].[8]

Function written versus function used: We believe implementation details must not effect the metrics. For example, the use of macros (with multiple in-line expansions) versus subroutines should not affect the metric value. Separating the portion of a product actually written from the portion created by a tool makes comparisons difficult; the standard productivity metrics do not work.

To meet this need, we developed the *Used Instruction*, (UI) [15]. We calculate the UI from the total function delivered to the customer, reported as if we wrote the entire software product as in-line code (no macros, inheritance, 4GLS, etc.). The difference between the "function written" and the "function used" indicates how well we program. The ratio of "function written" to RSI, or "function not written," indicates the level of reuse.

LOC we would not have written: Reusable parts often contain additional code to make the part more general. We decided that the cost of identifying these occurrences and the relatively small amount of code involved made it acceptable to count what we would not have normally written.

Definition of "ship": We report software metrics effective as of the product "ship date." However, although this term pertains to *external* customers, we also use the metrics for internal products.

When to report metrics: To provide in-process feedback to developers, we encourage metrics if identified as "in-process." This way reviewers understand the

[7] Software development organizations vary, but for measuring reuse we define a typical organization as either a programming team, department, or functional group of about eight people or more.

[8] For assessing the level of reuse in object-oriented projects we base the portion of reuse on the portion of reused classes. Our internal OO community not only recommends this unit but we do not always have more fine-grained data (e.g., the portion of methods reused) [11].

number may change and we can still monitor progress on products that may take years of planning and development.

Reporting by quality of reuse: IBM defines three levels of quality for reusable parts "as-is," "complete," or "certified"[19] . Like IBM, companies that certify parts may encourage reuse of these parts and identify needed parts by optionally reporting by quality level.

Availability of data: We must have easy data to collect, normalize, and report. Where data does not exist, we must make decide either to make the investment to obtain it or to use an alternative strategy.[9] We calculate the IBM reuse metrics from the following observable data elements used in IBM and other companies [10], [15]:

- Shipped Source Instructions (SSI)
- Changed Source Instructions (CSI)
- Reused Source Instructions (RSI)
- Source Instructions Reused by Others (SIRBO)
- Software Development Cost (Cost/LOC)
- Software Development Error Rate (Errors/LOC)
- Software Error (Cost/Error)

Combining the data: To develop metrics that reasonably reflect the levels of reuse, we carefully defined "reuse." We then asked:

1. Tell me the current level of reuse.
2. Tell me the value of the reuse, in dollars.
3. Tell me how I encourage building reusable parts.

We developed three reuse metrics to answer these questions [26], [27], [28], [30]. The first two metrics indicate the level of reuse activity in an organization as a percentage of products and by financial benefit. The third metric includes recognition for writing reusable code. Table 2 summarizes the three metrics.

"Rolling-up" the numbers: Options consist of:

1. *Vary organizational boundary.* Higher level managers only count parts from outside their organization. Although they may have high levels of reuse within the organization, the level may shrink when rolled-up.
2. *Average of all sub-organizations.* Although easy to compute, this option could mislead because it does not consider the relative efforts of the sub-organizations.
3. *Sum SSI and RSI.* IBM uses this option because it provides a weighted average of the levels of reuse in each of the sub-organizations.

Table 2. Derived Metrics

Metric	**Symbol**	**Derived from:**	**Unit of Measure**
Reuse Percent • for products • product releases • organizations	Reuse%	 • SSI, RSI • CSI, RSI • SSI, RSI	Percent
Reuse Cost Avoided	RCA	SSI or CSI, RSI, Cost/LOC, Errors/LOC, Cost/Error	Dollars
Reuse Value Added	RVA	SSI, RSI, SIRBO	Ratio

4.0 Conclusion

This paper discusses the issues surrounding development and implementation of reuse metrics; we required responsible and equitable resolution of the issues for their acceptance. Technical leaders demand responsible solutions because they insist on realistic and accurate values upon which to form their development decisions. Managers demand equitable solutions because they get judged and compared based on performance and results. These issues define the rules for such comparisons.

This paper has no known equivalent in the attention it gives to the definition of RSI and in attempting to present reuse as "real effort saved." Although [3] differentiates between reuse within and external to the organization, no other group provides a detailed explanation of how they count reuse.

5.0 Cited References

[1] Albrecht, A.J., and J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE TSE*, SE-9, 1983, pp.639-648.

[9] For example, in the case where a site collects process data by function point and not LOC, we replace LOCs in the metric with function points so they report reuse by "reused function points."

[2] Albrecht, A.J, "Measuring Application Development Productivity," *Proc. Joint IBM/ SHARE/GUIDE App. Symp.*, Oct.1979, pp.83-92.

[3] Banker, R.D., R.J. Kauffman, C. Wright, and D.Zweig, "Automating Output Size and Reusability Metrics in an Object-Based Computer Aided Software Engineering (CASE) Environment," *unpublished*, 25 Aug.1991.

[4] Barns, B.H. and T.B. Bollinger, "Making Reuse Cost Effective," *IEEE Software*, V.8, N.1, Jan.1991, pp.13-24.

[5] Bollinger, T.B., and Pfleeger, S.L, "Economics of reuse: issues and alternatives," *Inf. SW Tech.*, V.32, N.10, Dec.1990, pp.643-52.

[6] Basili, V.R., and D.M. Weiss, "A methodology for collecting valid software engineering data," *IEEE TSE*, SE-10, 1984, pp.728-738.

[7] Boehm, B.W., "Improving Software Productivity," *IEEE Computer*, 20, 1987, pp.43-57.

[8] Bourland, D.D. and P.D. Johnston, ed., To Be or Not: An E-Prime Anthology, *Int. Soc. for General Semantics*, SF, CA, 1991.

[9] Dabin, "Software Reuse and CASE tools," *Proc. Int. Comp. SW&Apps. Conf.*, 1991.

[10] Daskalantonakis, M.K., "A Practical View of Software Measurement and Implementation Experiences within Motorola," *IEEE TSE*, V.18, N.11, Nov.1992, pp.998-1010

[11] Devries, P., "Object Oriented Metrics," *IBM Doc. (draft V1.0)*, Dec.23 1992.

[12] Dreger, J.B. Function Point Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[13] Favaro, J., "What price reusability? A case study," *Ada Letters*, V.11, N.3, Spring 1991, pp.115-24.

[14] "Software Engineering Strategies," *Strategic Analysis Report*, Gartner Group, Inc., Apr.30, 1991.

[15] "Corporate Programming Measurements (CPM)," V4.0, *IBM Int. Doc.*, 1 Nov.1991

[16] Gaffney, J.E. and R.D. Cruickshank, "A General Economics Model of Software Reuse," *Proc. Int. Conf. SW Eng.*, Melbourne, 11-15 May 1992, pp.327-337.

[17] Firesmith, D.G., "Managing Ada projects: the people issues," *Proc. TRI Ada'88*, Charleston, WV, 24-27 Oct.1988, pp.610-19

[18] Humphrey, W., "Characterizing the Software Process: A Maturity Framework," *IEEE Software*, Mar.1988, pp. 73-79.

[19] "IBM Reuse Methodology: Qualification Standards for Reusable Components," *IBM Doc. No. Z325-0683*, 2 Oct.1992.

[20] Kolton, P. and A. Hudson, "A Reuse Maturity Model," *Pos. Paper, 5th Ann. WS on SW Reuse*, Center for Inn. Tech., Herndon, VA, 18-22 Nov.1991.

[21] Krueger, C.W., "Software Reuse," *Computing Surveys*, V.24, N.2, Jun.1992, pp.131-83.

[22] Margano, J., and L. Lindsey, "Software Reuse in the Air Traffic Control Advanced Automation System," *Joint Symp.&WorkShops: Improving the SW Proc. and Comp. Pos.*, Alexandria, VA, 29 Apr-3 May 1991.

[23] McCullough, P., "Reuse: Truth or Fiction," *panel position*, OOPSLA 92, Vancouver, CAN, 18-22 Oct.1992.

[24] Mukhopadhyay, T. and S. Kekre, "Software Effort Models for Early Estimations of Process Control Applications," *IEEE TSE*, V.18, N.10, Oct.1992, pp.915-924.

[25] Poulin, J.S. and W.E. Hayes, "IBM Reuse Methodology: Measurement Standards," *IBM Corp. Doc. No. Z325-0682*, 2 Oct.1992.

[26] Poulin, J.S. "Measuring Reuse," *Proc. 5th Int. WS on SW Reuse*, Palo Alto, CA, 26-29 Oct.1992.

[27] Poulin, J.S. and J.M. Caruso "A Reuse Measurement and Return on Investment Model," *Proc. 2nd Int. WS on SW Reusability*, Lucca, Italy, 24-26 Mar.1993, pp.152-166.

[28] Poulin, J.S. and J.M. Caruso, "Determining the Value of a Corporate Reuse Program," *Proc. IEEE CS Int. SW Metrics Symp.*, Balt., MD, 21-22 May.1993.

[29] Poulin, J.S., "Rationale and Criteria for the IBM Reuse Metrics," *IBM Tech. Report TR 00.3726*, 15 Apr.1993.

[30] Poulin, J.S., D. Hancock and J.M. Caruso, "The Business Case for Software Reuse," to appear, *IBM Systems Journal*, V.32, N.4., 1993.

[31] Prieto-Diaz, R., "Domain Analysis for Reusability," *Proc. COMPSAC '87*, 1987, pp.23-29.

[32] Reifer, D.J., "Reuse Metrics and Measurement- A Framework," *NASA/Goddard 15th Ann. SW Eng.WS*, 28 Nov.1990.

[33] Tausworthe, R.C, "Information models of software productivity: limits on productivity growth," *Jour. Sys. & SW*, V.19, No.2, Oct.1992, pp.185-201.