A Method for Assessing Cross-Lifecycle Reuse

Jeffrey S. Poulin

IBM Federal Systems Company
MD 0220, Owego, NY 13827
Tel: (607) 751-6899
Email: poulinj@vnet.ibm.com
Fax: (607) 751-2800

Abstract

Most current reuse measurement models focus on quantifying the level of software, e.g., code, reused on a given project [1], [2], [3]. One reason for the focus on code reuse metrics lies in the relative ease of quantifying the amount of code in a given product. A second, perhaps more critical, reason lies in the difficulty of completely understanding the issues behind reuse in other phases of the lifecycle, such as design [4], information [5], and test cases. This position paper describes a method to quantify the total amount of cross-lifecycle reuse on a project. However, until we have practical and reliable methods of understanding and quantifying the issues in each phase of software development, we will depend on our current code reuse models to reflect the overall level of reuse in our systems.

Keywords: Reuse Metrics, Measuring Reuse, Cross-lifecycle metrics.

Workshop Goals: Learn and exchange information on current reuse issues, methods, and metrics.

Working Groups: Design guidelines for reuse, Useful and collectible metrics, Reuse and formal methods.

1 Background

As a member of IBM's Reuse Technology Support Center, the author helped lead the development and acceptance of the IBM software reuse metrics. He has conducted research into software measurement techniques and implemented a program measurement tool for the workstation platform. A common aspect of this measurement work has remained the need to implement metrics that the software development manager can easily acquire, understand, and use with the confidence that the values upon which he makes his business decisions accurately reflect the state of his product.

2 Position

To fully determine the complete value of reuse, we must develop a method of measuring reuse in the total software *development* lifecycle. (The emphasis on development means to exclude maintenance activities.) The other phases of the lifecycle may include the reuse of requirements specifications, designs, user documentation, and other products related to the project. Since code really only contributes about 20 percent of the effort to software development [6], it seems misleading to claim a total level of reuse on a product based solely on the amount of reused source instructions.

However, we do not currently have a means to determine the total reuse on a product based on reuse in each of the lifecycle stages. We know that where code reuse takes place, reuse of designs, test cases, and documentation has probably also taken place. This "inclusion effect" [7] of reusing subsequent lifecycle products motivates us to reuse early in the lifecycle so as to generate the most value out of our reused parts. Because of this effect, I take the position (in the form of a hypothesis) that until we develop a comprehensive system of measuring and combining reuse in each phase of the software development lifecycle:

HO: total cross-lifecycle reuse $\approx$ level of code reuse

Observations show Reuse Percent (Reuse%) [8] as the most widely used and reported reuse metric. Reuse% has an advantage in that managers can easily gather data for, calculate, and understand the metric. This position paper assumes a systematic method of determining what to count (a reuse definition) [9] and does not attempt to address the financial aspects of cross-lifecycle reuse. Instead, it proposes to use (code) Reuse% as an overall indicator of the level of reuse on a product. The premise of this position extends the observation in [10] that despite their shortcomings, LOC not only serve as a good indicator of overall productivity in code but also provide a good secondary indicator of work done in other phases.

2.1 Proposed method to quantify total lifecycle reuse

While we use the code Reuse% as an overall product indicator, we can work towards realizing the following model of Product Reuse%. First , we require a means to measure reuse in each individual stage of the lifecycle. A straightforward approach to measure reuse in each phase would extend the approach currently used to measure reuse of code. First, determine the unit of granularity or workproduct unit of interest at each stage. Second, base the Reuse% for each stage on the portion of units reused in that stage. We might use the following units for each stage:

- **Code**– Lines of code, Function Points
- **Design**– Module design sheet, Component specification
- **Test**– Test cases, test scripts
- **Information Development**– Words, tokens, paragraphs

Observe that once we define the units to use at each phase and we put a process (and required tools) in place to track these units, we can determine the reuse percent for each phase. This may prove useful as an *in-process* measurement to indicate how well a product tracks towards a reuse target or goal. Also observe that this approach assumes each lifecycle phase has well-defined boundaries (e.g., we can calculate a Reuse% for each phase) and that the software development process remains relatively stable (e.g., that we can model the total lifecycle as the sum of the activity in each of the individual phases). Although these assumptions may not hold for any given project, they allow us to develop a general equation for Product Reuse%.

To achieve an overall Product Reuse%, we determine the portion of total product effort expended in each phase of the lifecycle. For post-mortem statistics, actual values will yield the best results, but for a general model we can use historical averages obtained from the organization product development office. For example, we might use the following lifecycle effort profile in our general model [6]:

- **Requirements Definition**– takes 15 percent of the lifecycle
- **Design**– takes 15 percent of the lifecycle
- **Code**– takes 20 percent of the lifecycle
- **Test**– takes 30 percent of the lifecycle
- **Information Development/admin**– takes 20 percent of the lifecyle

Knowing the Reuse% in each phase and the relative effort expended in each phase, we can calculate the overall Product Reuse% as follows:

$$\text{Product Reuse\%} = \sum_{i=1}^{n} (\text{Relative lifecycle effort in Phase}_i) \times (\text{Reuse\% in Phase}_i)$$

where n=number of phases in lifecycle.

2.2 Example total lifecycle reuse calculation

Let the entire software development cycle for an organization consist of the five phases above, in the proportions given. Let the following situations determine the amount reuse in each phase:

- **Requirements Definition**– 25 percent of requirements for the current system came from a similar system done last year for another customer

- **Design**– 55 percent of the designs, mostly coming from the requirements reused in the previous phase, came straight out of the graphical CASE tool library used by the design department,
- **Code**– 35 percent of the code came unmodified from the organization's reuse subdirectory on AFS,
- **Test**– 25 percent of the test cases and drivers from other projects worked just fine without change, and,
- **Information Development/admin**– 65 percent of the documentation, mostly from hypertext links to help files and user instructions, came from the Information Development (ID) database.

The Product Reuse% would equal:

- **Reuse%: Requirements Definition**– $.15 \times .25 = .038$
- **Reuse%: Design**– $.15 \times .55 = .083$
- **Reuse%: Code**– $.20 \times .35 = .07$
- **Reuse%: Test**– $.30 \times .25 = .075$
- **Reuse%: Information Development/admin**– $.20 \times .65 = .13$

For a total Product Reuse% of 39.6 percent.

3 Comparison

The detailed version of the COnstructive COst Model (COCOMO) software cost-estimation model uses phase-dependent Effort Multipliers to reflect the effect of the cost drivers on the phase distribution of effort [11]. The COCOMO model premises all calculations on cost drivers and Effort Multipliers that reflect the relative difficulty or complexity at each lifecycle phase. COCOMO defines the following six lifecycle phases: Requirements, Product Design, Detailed Design, Code and Unit Test, Integrate and Test, and Maintenance.

The model proposed by Gaffney and Durek [1] also addresses reuse at each lifecycle stage and uses effort multipliers similar to COCOMO. These models contrast with the model presented here, which does not consider the value or complexity of each task because this model does not seek to estimate costs or ROI. Were the model to address total lifecycle costs or costs avoided, it would extend the model presented in [2] by including a cost factor related to the value of the units we choose to measure at each phase.

References

[1] J. G. Jr. and T. Durek, "Software Reuse- Key to Enhanced Productivity: Some Quantitative Models," *Information and Software Technology*, vol. 31, no. 5, June 1989.

[2] J. Poulin and J. Caruso, "A Reuse Measurement and Return on Investment Model," in *Proceedings of the Second International Workshop on Software Reusability*, (Lucca, Italy), 24-26 March 1993.

[3] B. Barnes and T. Bollinger, "Making Reuse Cost Effective," *IEEE Software*, vol. 8, no. 1, pp. 13–24, January 1991.

[4] G. Boetticher, K. Srinivas, and D. Eichmann, "A Neural Net-based Approach to Software Metrics," in *Proceedings of the 5th International Conference on Software Engineering and Knowledge Engineering*, (San Francisco, CA), pp. 271–4, 14-18 June 1993.

[5] K. Yglesias, "Limitations of Certification Standards in Achieving Successful Parts Retrieval," in *Proceedings of the 5th International Workshop on Software Reuse*, (Palo Alto, California), 26-29 October 1992.

[6] G. G. Inc., "Software Engineering Strategies Strategic Analysis Report," tech. rep., April 30, 1991.

[7] T. Bollinger and S. Pfleeger, "Economics of reuse: issues and alternatives," *Information and Software Technology*, vol. 32, no. 10, pp. 643–52, December 1990.

[8] J. Poulin and J. Caruso, "Determining the Value of a Corporate Reuse Program," in *Proceedings of the IEEE Computer Society International Software Metrics Symposium*, (Baltimore, MD, 21-22 May 1993), pp. 16–27, 21-22 May 1993.

[9] J. Poulin, "Issues in the Development and Application of Reuse Metrics in a Corporate Environment," in *Fifth International Conference on Software Engineering and Knowledge Engineering*, (San Francisco, CA), pp. 258–262, 16-18 June 1993.

[10] R. Tausworthe, "Information models of software productivity: limits on productivity growth," *Journal of System Software*, vol. 19, no. 2, pp. 185–201, October 1992.

[11] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

4 Biography

Jeffrey S. Poulin, joined IBM's Reuse Technology Support Center, Poughkeepsie, New York, in 1991. His responsibilities on the RTSC included developing and applying corporate standards for reusable component classification, certification, and measurements. He currently works in Open Systems Development for IBM's Federal Systems Company and participates in the IBM Corporate Reuse Council, the Association for Computing Machinery, and the IEEE Computer Society. A Hertz Foundation Fellow, Dr. Poulin earned his Bachelors degree at the United States Military Academy at West Point, New York, and his Masters and Ph.D. degrees at Rensselaer Polytechnic Institute in Troy, New York.