2002

A Cost Benefit Model for Systematic Software Reuse

Marcus A. Rothenberger
University of Wisconsin - Milwaukee, rothenb@uwm.edu

Derek Nazareth
University of Wisconsin - Milwaukee, derek@uwm.edu

A COST-BENEFIT-MODEL FOR SYSTEMATIC SOFTWARE REUSE

Marcus A. Rothenberger; Derek Nazareth

School of Business Administration
University of Wisconsin – Milwaukee
PO Box 742
Milwaukee, WI 53211, USA
E-mail: {rothenb; derek}@uwm.edu

ABSTRACT

Information systems development is typically acknowledged as an expensive and lengthy process, often producing code that is of uneven quality and difficult to maintain. Software reuse has been advocated as a means of revolutionizing this process. The claimed benefits from software reuse are reduction in development cost and time, improvement in software quality, increase in programmer productivity, and improvement in maintainability. Software reuse does incur undeniable costs of creating, populating, and maintaining a library of reusable components. There is anecdotal evidence to suggest that some organizations benefit from reuse. However, many software developers practicing reuse claim these benefits without formal demonstration thereof. There is little research to suggest when the benefits are expected and to what extent they will be realized. For example, does a larger library of reusable components lead to increased savings? What is the impact of search effectiveness when evaluating reuse? This research seeks to address these questions. It represents the first step in a series wherein the effects of software reuse on overall development effort and costs are modeled with a view to understanding when it is most effective.

1. INTRODUCTION

Investments in information technology by U.S. businesses continue to spiral, with estimates topping $400 billion for 1997 (Strassman 1997). The explosive growth in the demand for software, coupled with shortages in the supply of software developers and the stagnant productivity in software development, has contributed to a perception of a "software crisis". The software development process in many organizations has been associated with cost and schedule overruns, and missing or erroneous functionality. Software reuse has been advanced as a means for easing this crunch. Potential benefits from software reuse include reduced development time and cost, improved software quality, increased developer productivity, greater sharing of knowledge/learning, improved maintainability of applications, easier adoption/enforcement of standards, among others.

Despite the potential rewards from an effective reuse program, it appears that widespread software reuse is not particularly prevalent. Different forms and extents of reuse are observed in the software development process, including ad-hoc reuse (which relies considerably on individual knowledge for reuse opportunities), planned reuse (often implemented through a reusable component library), to systematic reuse (embodying object-orientation for method level reuse), to inter-organizational reuse (as characterized by enterprise resource planning or industry-specific software).

There are a number of anecdotal reports describing the benefits of software reuse (e.g., Banker and Kauffman 1991). However, there is limited systematic exploration of the reuse phenomenon. Findings from one set of experiences with reuse are frequently not easily generalized to other settings. There is limited ability to answer questions like: Do larger projects benefit more from an established

reuse program? When does the reuse program start to pay off? Are the savings greater as the repository grows larger? Does module size affect the reuse savings? This research attempts to examine software reuse in a more systematic manner with a view to gaining a deeper understanding of the reuse phenomenon. To this end, a model investigating the impacts of software reuse in an organization is created. The model assumes the existence of a library of reusable software components, i.e. reuse is a planned philosophy. It examines the bottom line effects of component and program size in an effort to understand when reuse is a worthwhile endeavor.

2. SOFTWARE REUSE MODELS

Several software reuse models have been presented in the literature. Some provide means to quantitatively measure reuse benefit; others model the relationship between cost and benefit of reuse programs. For the purposes of this overview, we have classified the models into Reuse Level Metrics, Economic Models, and a category that is labeled "Other Models" for those literature contributions that don't fit the first two groups. This section briefly presents the most important existing models of each category.

2.1 Reuse Level Metrics

Banker and Kauffman (1991) developed a set of metrics in the context of a study that investigates the management of software reuse and the effect of the use of CASE technology at First Boston Corp. The metrics was introduced in the context of repository-based integrated CASE technology. It models the *Reuse Percentage* and the *Reuse Leverage*. Frakes and Terry (1994) have presented a measure that differentiates between internal and external reuse. It has been implemented as a tool that can calculate the metrics for a given systems. The Rothenberger and Hershauer (1999) measure implements a reuse level metric in the context of an enterprise-level data and process model environment that achieves reuse through a common architecture. It defines what type of components should count as reuse and argues that LOC can be a suitable complexity measure to assess the reuse rate.

2.2 Economic Models

Gaffney and Durek (1989) investigates how many times each component must be reused in order to pay off the effort invested in it. Hereby, it is assumed that the total cost of a new software system is equal to the sum of the costs of its new and reused components. The impact of reuse is measured relative to the effort required to develop the project from all-new code. The metric can help organizations to estimate development time and to make the trade-off between the proportion of reuse and the cost of developing and reusing components. Poulin and Caruso (1993) developed a model to improve measurement and reporting of software reuse. They present a measure for reuse level, the financial, and the productivity benefit of reuse. Also, they provide two return-on-investment models for analysis of reuse on the project, as well as on the corporate level. Barnes and Bollinger (1991) provide an analytical approach for making good reuse investment decisions. They present a model that relates the reuse benefits to the cost of reuse and then they analyze several methods of improving the cost-effectiveness of reuse based on the metric introduced. Assuming that an organization can measure or estimate cost and benefit of reuse, the number of projects necessary for reuse to pay off can be calculated.

2.3 Other Models

The Balda and Gustafson (1990) model allows estimating the software project cost with reuse based on the COCOMO Model (Boehm 1981). Their model takes into account that additional effort required

to develop components for reuse, as well as the effort savings of building applications from reusable software artifacts. The model allows organizations to estimate the development effort of a project that is to be developed using both, reused and newly written components. The project productivity in a reuse-based development environment can be assessed with a model presented in (Rothenberger and Dooley 1999). This paper develops productivity measures for issues that only apply to a reuse development context, such as the quality of the reuse decision and the efficiency of component retrieval. These single measure then are integrated into a comprehensive reuse project productivity metric.

Existing reuse models focus on particular aspects of reuse in the software development cycle (Poulin 1997). The models provide software development groups with a tool to institutionalize reuse. This means that in order make software reuse a regular integrated part of the software development process, reuse models and metrics can help the organizations to achieve their reuse goals. Reuse models provide a standard that decides what counts as reuse and how is it measured or to assess cost savings that are to be attributed to the success of reuse. Accordingly, some models provide means to estimate the cost of reuse projects, or required reuse occurrences to break even; while most of the models discussed are an ex-post analysis of the quality or success of reuse that helps organizations to decide whether to continue the reuse approach as is or whether to change directions.

In the literature, there is ample anecdotal evidence that describes reuse success stories (Apte, Sankar et al. 1990, Banker and Kauffman 1991, Lim 1994, Poulin, Caruso et al. 1993). Unfortunately these organizations represent a small group of businesses that managed to obtain great benefit from their reuse programs. Many competitors have implemented similar programs, just to realize that reuse did not deliver the expected paybacks to their development process (Frakes and Fox 1996). Existing models can help companies to assess the success of their reuse programs. They might even help to improve reuse methods that are in place. However, they don't tell organizations ahead of time how to best implement reuse into the development process. The adoption of a reuse program would be less costly and more successful, if researchers could provide a guideline to potential adopters describing how reuse is implemented best.

Existing models can only address limited-scope questions; each focuses only on a particular aspect of reuse (Poulin 1997). For example, the Reuse-Level Metrics models are concerned with how much reuse is achieved and the Economic Models answer how many projects must be developed to reach a break-even point after the introduction of a reuse program. The literature has not presented a comprehensive model of reuse that includes all elements of the reuse process and their interaction with one another. A comprehensive view is crucial in order to utilize a model in a predictive manner and answer the questions above.

In this research, such a comprehensive cost-benefit model is developed to move closer towards answering above-mentioned questions about reuse.

3. A DOMAIN-SPECIFIC COST-BENEFIT MODEL

Modeling the effectiveness of a library of reusable components necessitates some assumptions as to the scope and operation of the reuse phenomenon. In this research, we assume that the library addresses software modules that support a specific domain. A domain is characterized as a set of information systems that possess similar functionality and share the same underlying data. Typically a domain will address related processes and involve a limited set of users. Domains frequently break down along departmental or divisional lines within an organization. It is expected that reuse potential is far greater within a domain than across domains. This is understandable given the close ties between processes supported and the overlap in users and data. Thus for example, there will likely be little reuse potential between an order management system and a payroll system, other than common

infrastructure support. On the other hand, the payroll system and a personnel inventory system are likely to offer greater opportunity for reuse.

The model breaks down software reuse costs along dimensions of repository creation, search for reusable modules, modification of software modules to fit the specified use, and contrasts these with the development costs assuming no reuse.

3.1 Search costs

Search costs are composed of two components – query costs and retrieval costs. Query formulation costs are based on the number of terms to be retrieved, moderated by the effectiveness of selecting among the query criteria. Retrieval costs are based on the number of components to be searched, the number of query criteria, the selectivity among criteria, and the effectiveness of retrieval.

Retrieval costs using a fixed selectivity are underestimated as compared to the costs that are incurred when different selectivities are employed for each search criteria. An analysis of the effect of different selectivities indicated an underestimation of approximately 20% for each new criterion added into the search.

3.2 Publication Costs

Publication costs represent the costs associated with the development of new components, any modification costs associated with customizing the components for use in a project, costs associated with making a component generic, as well as cataloging costs. Development costs are computed based on the complexity of the component (in this case modeled as component size), moderated by the relative effectiveness for new code development. Modification costs are modeled using the quality of the retrieved component, the size of the component, and the relative effectiveness of modification. Making a component generic is a function of the quality of the retrieved component. Cataloging costs are modeled simply on the basis of the number of cataloging dimensions, the fraction of components reused "as is", and the cataloging effectiveness. Note that not all components will involve development – just those where there is no appropriate match. Likewise, only some modules will need to be made generic – based on a desired threshold for quality. All costs are for a single component only. Clearly project size will influence the overall publication costs, based on the proportion of components that can be reused "as is", the proportion of components that can be modified, the proportion of components that need to be made generic, and the proportion of components that must be generated because no reuse is possible.

These represent per component costs. Project costs would factor in the number of components in the project and the reuse rate for the components. Development cost savings is computed as a simple difference between overall cost for the project assuming reuse and development costs assuming no reuse at all.

3.3 Calibrating the model

The effectiveness of the model is largely determined by its robustness and its reasonableness. In an effort to ensure that the model produced reasonable results, the various proportions for reuse, reuse as is, and the degree of fit of the retrieved component *(r, p, s)* were calibrated to be a function of total repository size. The proportion of components reused *r,* is bounded by a maximum value r_{max}, representing the best that the enterprise can achieve in terms of reuse. In addition, the enterprise may adopt a strict or liberal reuse policy. The former dictates that reuse occurs only if there is no rework effort, while the latter allows for modification of retrieved components, provided that the modification effort is manageable. Note that a strict reuse policy entails that the proportion of components reused

as is *p,* will necessarily be 1, and the modification costs will necessarily be 0. Likewise, the degree of fit *s,* will also increase as the number of components in the repository increase, subject to a minimum s_{min}, representing the point at which it is deemed that the modification effort precludes effective reuse. These relationships are depicted in Figure 1.

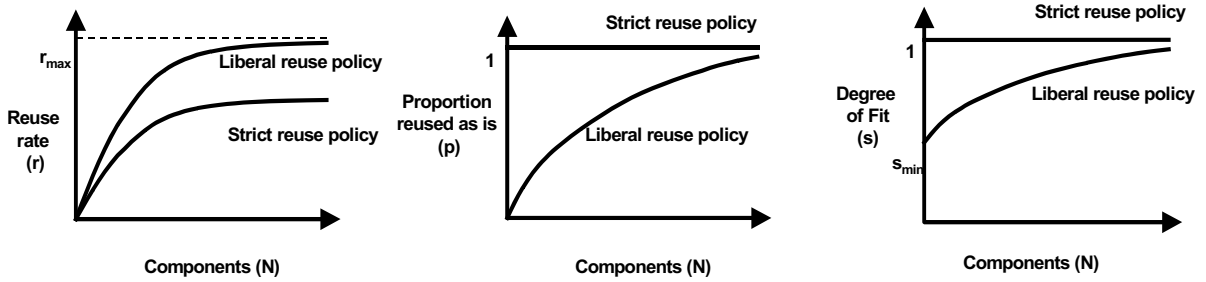

Figure 1. Model Calibration Properties

3.4 Sensitivity to Parameter Changes

The relationships introduced in the model are functions of various constants. In order to fine-tune the model, the values of the constants must be approximated. This section will demonstrate that the model is rather insensitive to some constants, while it is more sensitive to others. Only for the latter it is crucial to correctly estimate their values. *Modification Time per Complexity Unit* (M_E) and the *Time Needed to Make Code Generic per Complexity Unit* (G_E) are not sensitive to changes, while the *Development Time per Complexity Unit* (D_E) is sensitive. This is, because in case the development cost is insignificant, it is not worthwhile for an organization to incur the overhead necessary for a reuse program. If, however, the development cost is high, then reuse can pay off.

The non-sensitive variables M_E and G_E can be roughly estimated. It would not affect the analysis even if the estimates of those values were not exact. The estimate for D_E requires a higher accuracy in order to avoid falsifying the results. Many researchers have attempted to estimate G_E. Table 1 summarizes their findings.

Poulin (1997) has discussed these publications. He draw the conclusion to use the median of all literature-identified values is the best estimate for G_E. According to this, writing generic code for reuse takes 1 ½ times the time than writing the same code for one specific application. Hence, $G_E = ½ * D_E$. It must be noted that the true value of G_E depends on various factors inherit to the organization, such as development environment, skill of developers, and domain of the application. We feel, that the median of these empirical studies best represents a typical value for G_E, which allows us to draw conclusions about reuse in general. Hence, we shall use this estimate for the remainder of the analysis.

Publication	***Estimate for*** G_E	***Comments***
Margano and Lynn, 1992	$G_E = D_E$	in the context of aviation applications
Favaro, 1991	$G_E = 0$ to $G_E = 1.2 * D_E$	
Lenz, et al., 1987	$G_E = .25 * D_E$ to $G_E = D_E$	
Lim, 1994	$G_E = .11 * D_E$ to $G_E = .8 * D_E$	
Reifer, 1990	$G_E = .1 * D_E$ to $G_E = .2 * D_E$	
Bardo, et al., 1996	$G_E = .15 * D_E$ to $G_E = .25 * D_E$	based in estimates only
Poulin, 1995	$G_E = .86 * D_E$	
Pant, et al., 1996	$G_E = .55 * D_E$	

Table 1: Literature Estimates for G_E

3.4.1 Savings

As the Development Cost goes up, Savings increase. Figure 1 demonstrates that for constant Development Cost, the savings increase as the number of components increase. However, the savings increase levels off, which indicates that there is a diminishing return on increasing the repository. The same holds for an increase for the component complexity. As the complexity increases, the savings grow, however, the increase levels-off beyond a certain complexity.

3.4.2 Startup Cost

As the Development Cost goes up, the Startup-Cost goes up as well. The diagram (Figure 1) illustrates that a reuse program will accrue losses when the repository is too small. As components are developed from scratch and added to the repository, the losses are reverted into gains. The minimum number of components required to break even depend on the Development Cost. As the Development Cost goes up, the minimum number of components required in the repository decreases. Larger repositories then will result in savings.

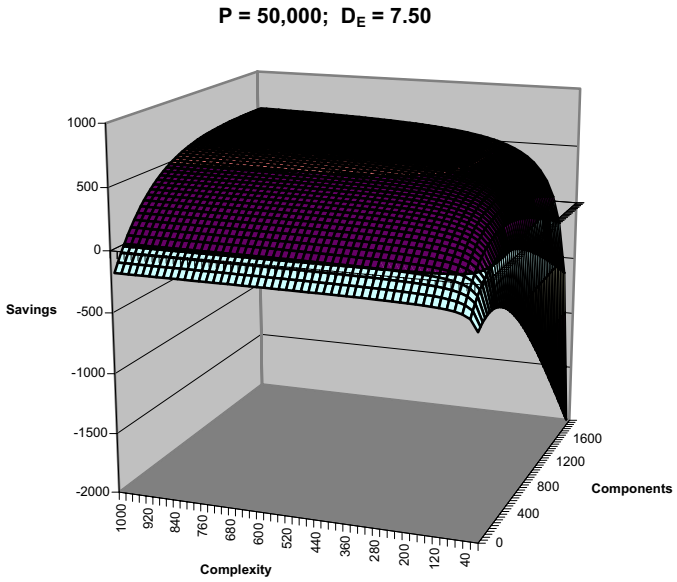

Figure 2: Model Plot

4. CONCLUSION

The model enables us to answer the questions that were raised in the introduction. In this section we will answer the questions based on the insights obtained through the relationships that were established.

4.1 Are the savings greater as the repository grows larger?

The Model Plot (Figure 1) illustrates that the savings level out, once a specific number of components in the repository are exceeded. That means that adding additional components will only increase the

cost, but not the benefit from reuse. Hence, we conclude that there is an optimal repository size for a reuse environment. In the particular case depicted in Figure 1, we are looking at a project of 50,000 complexity units and a development productivity of 7.5 complexity units per standard time unit. In this scenario, the leveling off occurs beyond approximately 1,000 components in the repository.

4.2 Does module size affect the reuse savings?

The x-axis on the diagram marks the average component complexity. In is apparent that only a very small average component complexity can hurt reuse (in this scenario, less than 200 complexity units per component). Once this number is exceeded, additional benefit cannot be obtained by increasing component size. The diagram shows that savings level off for larger average module size. While the reuse savings remain the same, increasing component size will lead to less flexibility in meeting application requirements. This will require a narrowing of the target domain. The aspect of development flexibility is not modeled and therefore does not appear on the diagram.

4.3 Do larger projects benefit more from an established reuse program?

The relationships used in the model also indicate that the cost savings from reuse are linearly related to the project size, if the average component complexity and the component size remain unchanged.

4.4 When does the reuse program start to pay off?

Further, an organization that starts with a 0-size repository will accrue an increased development cost until the repository contains a sufficient number of components. An organization pursuing this adoption route would populate the repository over time by making components generic that were written for applications when no reuse opportunities were found. Making components generic for reuse will require additional development time, while not providing a benefit in savings for the initial projects. According to the diagram, it may be more efficient to start a reuse program with a repository that is populated with a sufficient number of components that warrants cost savings. In the scenario depicted in Figure 1, the reuse program pays off for repositories of more than 100 components (assuming a complexity per component of more than 200 complexity units).

4.5 Summary

This research formulated a model for software reuse, noting the explicit costs and benefits associated with a reuse program. Running the model for various scenarios has provided insight into the benefits that can be expected as part of a reuse program, permitting more definitive answers to the managerial questions that normally accompany the creation or proposal of a reuse program. In summary, we have learned that an organization must accept initial losses on reuse while the repository is being populated; later, the repository must not expand indefinitely, as an optimal repository size exists. Further, reuse benefits are greater from more complex reusable components; however, this positive effect levels off beyond a certain component size. Further, we have learned that project size does not matter. Large projects benefit equally from reuse as small project, measured in percentage of total effort. These insights may affect the planning of the adoption of new reuse programs.

REFERENCES

Apte, U., C. S. Sankar, et al. (1990). "Reusability-Based Strategy for Development of Information Systems: Implementation Experience of a Bank." MIS Quarterly **14**(4): 420-433.

Banker, R. D. and R. J. Kauffman (1991). "Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study." MIS Quarterly **15**(3): 374-401.

Barnes, B. H. and T. B. Bollinger (1991). "Making Reuse Cost-Effective." IEEE Software **8**(1): 13-24.

Boehm, B. W. (1981). Software Engineering Economics. Englewood Cliffs, NJ, Prentice Hall.

Frakes, W. B. and C. J. Fox (1996). "Quality Improvement Using A Software Reuse Failure Modes Model." IEEE Transactions on Software Engineering **22**(4): 274-279.

Frakes, W. and Terry (1996), C., Software Reuse: Metrics and Models, ACM Computing Surveys, **28** (2): 415-435.

Gaffney, J. E. and T. A. Durek (1989). "Software reuse - key to enhanced productivity: some quantitative models." Information and Software Technology **31**(5): 258-267.

Kim, Y. and Stohr (1998), E.A. Software Reuse, Journal of Management Information Systems, **14** (4), 113-147.

Krueger, C.W. (1992), Software Reuse, ACM Computing Surveys, **24** (2), 131-183.

Lim, W. C. (1994). "Effects of Reuse on Quality, Productivity, and Economics." IEEE Software **11**(5): 23-30.

Lim, W.C. (1996), Reuse Economics: A Comparison of Seventeen Models and Directions for Future Research, in Proceedings of the Fourth International Conference on Software Reuse, (M. Sitaram, ed.), IEEE Computer Society Press, Los Alamitos, California, 41-50.

Poulin, J. S. (1997). Measuring Software Reuse: principles, practices, and economic models. Reading, MA, Addison-Wesley.

Poulin, J. S. and J. M. Caruso (1993). A Reuse Measurement and Return on Investment Model. 2nd International Workshop on Software Reusability, Lucca, Italy.

Poulin, J. S., J. M. Caruso, et al. (1993). "The business case for software reuse." IBM Systems Journal **32**(4): 567-594.

Radding, A. (1998), "Hidden costs of reuse", InformationWeek, Manhasset; Nov 9, Iss. 708; pg. 1A, 5 pgs.

Rothenberger, M. A. and K. J. Dooley (1999). "A Performance Measure for Software Reuse Projects." Decision Sciences **30**(4).

Rothenberger, M. A. and J. C. Hershauer (1999). "A Software Reuse Measure: Monitoring an Enterprise-Level Model Driven Development Process." Information & Management **35**(5): 283-293.

Strassman, P. (1997), *The Squandered Computer: Evaluating the Business Alignment of Information Technologies*, Information Economics Press, New Canaan, CT.