01 Feb 2005

A Graph-Based Model for Component-Based Software Development

Sahra Sedigh
Missouri University of Science and Technology, sedighs@mst.edu

Arif Ghafoor

Recommended Citation

S. Sedigh and A. Ghafoor, "A Graph-Based Model for Component-Based Software Development," *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (2005, Sedona, AZ)*, pp. 254-259, Institute of Electrical and Electronics Engineers (IEEE), Feb 2005.
The definitive version is available at https://doi.org/10.1109/WORDS.2005.8

A Graph-Based Model for Component-Based Software Development

Sahra Sedigh-Ali
University of Missouri-Rolla
Department of Electrical and Computer Engineering
Rolla, MO, 65409-0040 USA
sedighs@umr.edu

Arif Ghafoor
Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
ghafoor@ecn.purdue.edu

Abstract

Software metrics can be used to objectively quantify the quality of software components and systems, alleviating quality and risk concerns and raising assurance in component-based systems. In this paper, we present a graph-based model for component-based software development. We assume that a number of components have been characterized in terms of non-functional metrics of importance to the software system being developed, and that the interfaces connecting various components have been similarly characterized. The emphasis of this work is on cost and quality of the system under development, and reaching an acceptable compromise between the two.

1. Introduction

The paradigm shift to *commercial off-the-shelf* (COTS) components appears inevitable, necessitating drastic changes to current software development and business practices [6]. Quality and risk concerns currently limit the application of *COTS-based software* (CBSs) to non-critical applications [29]. New approaches to quality and risk management are needed to handle the growth of CBSs [4, 28]. With software development proceeding at unprecedented speed, in-house development of all system components may prove too costly in terms of both development resources and time, as witnessed by the outsourcing trend currently present in commercial software development. Large-scale component reuse and COTS component acquisition can generate savings in development resources, which can then be applied to quality improvement, such as enhancements to reliability, availability, and ease of maintenance.

In this paper, we present a graph-based model for component-based software development (CBSD). We assume that a number of components have been characterized in terms of non-functional metrics of importance to the software system being developed, and that the interfaces connecting various components have been similarly characterized. The emphasis of this work is on cost and quality of the system under development, and reaching an acceptable compromise between the two. The objective of component selection is to find a set of components that meets all functional requirements of the system while achieving the best tradeoff between non-functional metrics.

The remainder of this paper is organized as follows. Section 2 provides a summary of related literature. In Section 3, we present a graph-based model for the optimization of CBSD, and formulate the selection of components as an optimization problem. We conclude the paper by summarizing the findings in Section 4.

2. Related Work

Considerable research has been conducted in software metrics in the past decade, especially on reliability models, cost estimation, and application of software metrics [7, 8]. The bulk of this research is restricted to traditional, not component-based software systems. A framework for cohesion measurement in object-oriented systems has been used

[0]This research was supported by the National Science Foundation under Grant No. EIA-9972883, and by the e-Enterprise Center at Purdue Discovery Park.

Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)
0-7695-2347-1/05 $20.00 © 2005 IEEE

in [5] to empirically explore relationships between design coupling, cohesion, and inheritance measures for object-oriented (OO) systems. Both studies, which are related to the heavily-cited "C and K" suite of OO metrics proposed in [9], focus on OO measures, and require at least class-level access to program code. This may prevent the applicability of the proposed metrics to COTS components only accessible through interfaces at the component level.

The cost and quality metrics of greatest interest to this paper are adapted from [24], which identifies a set of management, requirements, and quality metrics for component-based software development. The research in [22] investigates metrics-guided software reuse, which is related to quality management in our research, but the emphasis is on reuse decisions, not quality. This study served as a background for our research on metrics-guided quality management.

Very little, if any, research has been conducted on the economics of quality in CBS development. Cost models for software reuse have been widely studied, but quality is largely ignored in these studies. COCOMO 2.0 [3] takes software reuse into account, and allows the use of logical lines of code (LLOC) as the standard measure. This model has limited applicability to CBSs, as COTS software, libraries, and auto-generated code are excluded when counting the LLOCs. Where possible, COCOMO 2.0 can be used to estimate some component-level cost factors for CBSs.

The *Constructive COTS* (COCOTS) model [2], a COCOMO derivative, can be used to estimate effort and schedule for CBS development. Of relevance to our work is the assessment sub-model of COCOTS, which is intended for use in the initial stages of development, and is aimed at selecting the most suitable COTS component from a set of candidates. COCOTS can currently yield effort estimates only; schedule estimation is yet to be incorporated into the model.

Estimation of COTS integration costs has been performed in [13]. A cost model for software reuse is proposed in [19]. The research in [11] compares in-house development to COTS acquisition by adjusting costs to their net present value. All three studies perform cost-benefit analysis for software reuse, but none of them discusses the quality of the integrated system.

In [16], the reliability of a software system is estimated based on the reliability of its components. This study employs a "function call graph" to depict the interaction between components, similar to our graph-based CBS model. Contrary to our work, it does not assume serial execution of components. The reliability estimation performed is more general than our estimation, but is the only focus of the research. It does not address the estimation of other quality metrics or the development process.

The tradeoff between cost and reliability has been widely studied, and several formulations for optimizing software reliability have been proposed. In [12], system-level reliability has been evaluated as a function of component-level failure intensities, and the optimization problem seeks to minimize the total cost of achieving the desired reliability by allocating specific failure intensities to the components. This approach may be applicable to the development of similar formulations for other software metrics.

In [7], a method is proposed for estimating the development costs of a software module, taking into account the target reliability. A decomposition technique is used to estimate the cost of development, based on an estimate of the number of faults that need to be found and fixed in order to achieve the desired reliability. This model only considers the costs of coding and testing, and as such, does not provide a comprehensive method for allocating effort to quality initiatives. Other studies [1, 30] determine optimal release strategies based on reliability and cost criteria, but do not provide a unified approach to optimizing more than one component of quality.

Off-the-Shelf Option (OTSO) [15] is a method supporting the search, evaluation and selection of reusable software. This method relies on the use of multilevel programming in its optimization. For the reasons explained in Section 3, this technique is of limited applicability to situations where several of the criteria are of comparable importance. In addition, the OTSO method does not provide a metric set that can be used in the evaluation criteria, and does not explicitly determine the factors contributing to the quality of the CBS.

An optimization model for developing a modular software system is presented in [14]. This study shares a basic assumption with our work: the system is assumed to be comprised of a set of serially executed modules, where each module is configured with only one COTS product. The objective function of the model minimizes the system development cost. This study does not consider quality factors other than failure rate in the analysis. Due to the integer programming approach taken in solving the optimization problem, this method cannot be easily generalized to consider additional quality factors.

In [18], the optimization of CBSD is formulated in a manner similar to our formulation. Priorities are then specified for various non-functional attributes, and a bargaining game is used to find one Pareto-optimal configuration for the CBS. This approach is prone to the shortcomings described for the weighted sum method in Section 3.2, as it eliminates the insight gained by allowing a choice between multiple Pareto-optimal system configurations.

3. Optimization of CBSD

Metrics-guided quality management can involve formulating software development decisions as optimization problems, and solving the resulting problems. For example, for the metrics representing quality costs, quality, and complexity, a tradeoff can be achieved where a high-quality CBS is developed at low cost, without becoming overly complex.

We consider the main decision in component-based software development to be acquisition of the most suitable components. We assume that a *family of systems* (FOS) is available for each functionality desired for the software system. Each family provides a number of pre-packaged candidate components for incorporation into the CBS. For example, family A is composed of M software components (available alternatives), denoted by $\mathrm{A}_1, \mathrm{A}_2, ..., \mathrm{A}_M$. We assume that the functional attributes of each candidate in the family fulfill the functional requirements of the CBS; hence, the choice among candidate components is based solely on *non-functional* attributes. This assumption does not lead to any loss of generality, as a simple feasibility check can disqualify any candidates that do not meet the functional requirements. In this paper, the metrics of interest are reliability, complexity, and cost, due to their critical role in the viability of a commercial software system.

We are making two simplifying assumptions regarding the CBS:

1. The families of systems have non-overlapping functionality. Thus, exactly one component is chosen from each family to design the CBS.

2. The components are connected serially, i. e., each component is interfaced to at most two other components.

We maintain that neither assumption leads to a significant loss of generality. The first assumption can be justified by selecting components of suitable granularity. The second assumption is justified for systems where components communicate through shared objects in addition to their explicit interfaces, so the mathematical tractability gained by the simplification is not at the expense of a significant departure from reality. In related literature, [14] makes very similar assumptions in studying CBSs.

3.1. A Graph-Based Model for CBSD

Figure 1 depicts a graph-based representation of the component selection problem. The candidate components are represented as nodes. The components belonging to each family of systems are enclosed within a dashed line. The interface between two components is represented as an edge. As we are selecting at most one component from each FOS, there are no edges between candidates from the same family. Associated with each node are the component-level metrics of interest. In this paper, the metrics of interest are complexity, reliability, and acquisition cost, denoted by l, r, and c, respectively. Similarly, a number of metrics are associated with each edge; for this paper, we are interested in the integration cost, complexity, and reliability of the integration code. Integration code can be purchased as middleware or developed in-house. In either case it can be characterized by the three aforementioned metrics.

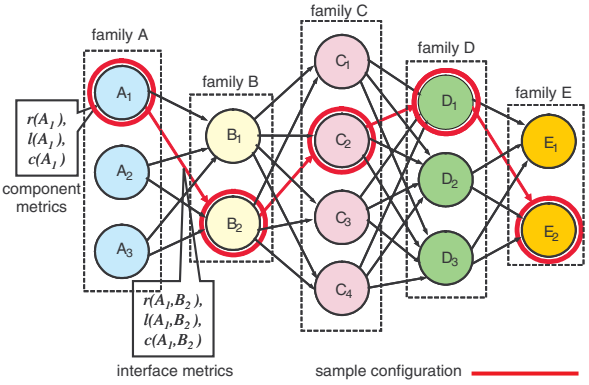

Figure 1. Graph-based representation of the component selection problem.

For both components and integration code, we are defining reliability, $r(x)$, as the probability of failure-free operation of the program or program segment x. Complexity, $l(x)$, is defined as the number of interacting modules in the component or integration code x. The cost, $c(x)$, associated with a component (node) x is acquisition cost, whereas the cost $c(x_i, x_j)$ associated with an edge (integration code) (x_i, x_j) represents the cost of acquiring integration middleware or developing the necessary integration code in-house.

We assume that the component vendor has provided specifications that include the needed metric values. In the absence of such data, the developers of the CBS need to measure the metric values by testing the component. The majority of this testing is black-box testing, as the source code of COTS components is typically inaccessible. Any metrics that cannot be measured by testing have to be estimated by simulation, or by extrapolating values from field use of prior versions or similar components from the same vendor. If the integration code is acquired as middleware, the same discussion applies to metric values for the edges of the graph. For in-house integration code, metric values are measured by white-box testing of the code.

A sample CBS configuration has been outlined in Figure 1. The ultimate objective of optimizing the CBSD process depicted in this figure is guiding the selection of such a sys-

tem configuration, which contains a single component from each family, along with the integration code required for interfacing the selected components. The problem can be considered an extension to the shortest path problem, where instead of the single "length" objective, multiple objectives are optimized for the path traversing all families.

3.2. Formulation of CBSD as an Optimization Problem

We assume that all of the candidate components and interfaces fulfill the functional requirements for their respective FOSs. The goal of the CBSD problem at hand is finding a system configuration that satisfies the following three objectives:

1. maximizing system reliability, $R(\mathbf{x})$, which corresponds to minimizing system risk,
2. minimizing system complexity, $L(\mathbf{x})$, and
3. maintaining system cost, $C(\mathbf{x})$, within budget limits.

In this context, a *system configuration* refers to a set comprised of one component from each FOS, along with the interfaces required for linking consecutive components. It is readily apparent that the objectives may conflict, as the most reliable system configuration may not be the least complex. This conflict of objectives is typical of many system design decisions. Such problems lend themselves to *multi-objective optimization* (MOO) techniques, which optimize a number of criteria simultaneously [27]. A multi-objective optimization problem (MOP), in its general form, can be formally stated as:

$$\begin{array}{rll} \text{Min/Max} & \phi_m(\mathbf{x}), & m = 1, 2, \ldots, M; \\ \text{s.t.} & g_j(\mathbf{x}) \geq 0, & j = 1, 2, \ldots, J; \\ & h_k(\mathbf{x}) = 0, & k = 1, 2, \ldots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \ldots, n. \end{array}$$

A solution $\mathbf{x}$ is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$. The bounds placed on $\mathbf{x}$ constitute the *decision variable space* $\mathcal{D}$, simply denoted as the decision space. $g_j(\mathbf{x})$ and $h_k((x))$ are *constraint functions*. A solution $\mathbf{x}$ must satisfy all $J + K$ constraints, as well as the variable bounds, to be considered a *feasible solution*. The goal of the problem is finding the best tradeoff among the set of M *objective functions*, $\{\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_M(\mathbf{x})\}$.

For component-based software development, each decision variable, x_i, in the vector $\mathbf{x}$ corresponds to one component of the system. The value of x_i corresponds to the choice of a particular candidate from an FOS. For a component-based system with n components, the decision vector is $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$. The objectives of the optimization can correspond to various system-level metrics of interest. In this paper, the focus is on reliability, complexity, and cost. The constraints for the problem can arise from system requirements, budget restrictions, or similar design considerations.

We have assumed that system reliability, $R(\mathbf{x})$, can be represented by the product of the component and interface reliability for every component and interface in the system. We further assume that complexity is additive, i. e., the overall system complexity, $L(\mathbf{x})$, can be obtained by adding the complexity of all components and interfaces in the system. We assume that system cost, $C(\mathbf{x})$, is similarly additive. These assumptions are very common in software engineering literature [12, 17, 21, 26].

Reliability is assumed to be multiplicative, as it represents the probability of failure-free operation. Complexity is assumed to be additive, as the total number of interfaces resulting from the integration of a number of software modules is no greater than the sum of the number of interfaces in the individual modules. The minimum acceptable value of reliability, maximum acceptable value of complexity, and maximum acceptable value of cost are denoted as $R_{\min}$, $L_{\max}$, and $C_{\max}$, respectively.

Using the above notations, the problem of optimizing CBSD for a system of n components is formally stated as follows:

$$\begin{array}{rl} \text{Min} & \phi_1(\mathbf{x}) = L(\mathbf{x}) = \sum_{i=1}^{n} l(x_i) + \sum_{j=1}^{n-1} l(x_j, x_{j+1}) \\ \text{Min} & \phi_2(\mathbf{x}) = -R(\mathbf{x}) = -\prod_{i=1}^{n} \prod_{j=1}^{n-1} r(x_i) r(x_j, x_{j+1}) \\ \text{s.t.} & g_1(\mathbf{x}) = 1 + \frac{\phi_1(\mathbf{x})}{R_{\min}} \leq 0 \\ & g_2(\mathbf{x}) = \frac{\phi_2(\mathbf{x})}{L_{\max}} - 1 \leq 0 \\ & g_3(\mathbf{x}) = \frac{\sum_{i=1}^{n} c(x_i) + \sum_{j=1}^{n-1} c(x_j, x_{j+1})}{C_{\max}} - 1 \leq 0 \\ & x_i^1 \leq x_i \leq x_i^{p_i},\ i = 1, 2, \ldots, n. \end{array}$$

The solution, $\mathbf{x}$, is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$, where each x_i represents the candidate chosen from FOS i, which is assumed to contain p_i candidate components.

As the scalar concept of "optimality" does not directly apply to the multi-objective setting, *Pareto-optimal* [23] solutions are found. A Pareto-optimal solution is one that cannot be improved with respect to any one objective without worsening some other objective; hence, the optimal compromise is reached between the various objectives. In contrast to scalar optimization problems, which typically have a single solution, MOPs generally have many Pareto-optimal solutions, which are collectively referred to as the *Pareto front*. Any solution in the Pareto front is by definition *non-dominated*. A solution $\mathbf{x}^{(1)}$ is said to dominate another solution $\mathbf{x}^{(2)}$ if $\mathbf{x}^{(1)}$ is no worse than $\mathbf{x}^{(2)}$ in all objectives, and is strictly better than $\mathbf{x}^{(2)}$ in at least one objective.

Formulating component-based software development as a MOP, and solving the MOP by finding the Pareto front,

yields a better understanding of the tradeoffs among various attributes of the system. One common technique for finding the Pareto front of MOPs involves combining the multiple objectives into one scalar objective whose solution is provably Pareto-optimal point for the original MOP. The most common example of this technique is the weighted sum method, where a positively weighted convex sum of the objectives, $\sum_{i=1}^{M} \alpha_i \phi_i(\mathbf{x})$, $\alpha_i > 0$, $i \in \{1, 2, ..., M\}$, is minimized [10]. For CBS quality management, the weights are chosen to represent the relative importance of the metrics. The selection of appropriate weights is critical to this approach.

In *multilevel programming*, which is another commonly used technique, the objectives are ordered according to the their relative importance. The optimization then begins by finding the optimizers of the first objective function and recursively proceeds to find the optimizers of all of the objective functions, using the optimizers of the previous step as the decision space for the current step. The study in [15] applies this method to the selection of COTS components in CBSD.

This approach is useful when the hierarchical order among the objectives is of prime importance, not the continuous tradeoff among them. However, objectives lower in the hierarchy become very tightly constrained and often numerically infeasible, so that the less important objectives wield no influence on the final quality. Hence, multilevel programming should be avoided in cases where a sensible compromise among the metrics is sought [20].

Our solution of choice is evolutionary programming, in particular, genetic algorithms. Our approach is described in [25].

4. Conclusions

This paper proposed a graph-based model for component-based software development. This model can guide the selection of components from a family of candidates, where the objective of the selection is to develop a system that satisfies given non-functional requirements. The model proposed guides this selection by identifying the components that will collectively achieve the best tradeoff among the metrics desired for the system. This technique results in a quality management method that can alleviate concerns regarding uncertainty in the cost and quality of a component-based system.

References

[1] N. Ashrafi and F. Zahedi. Software reliability allocation based on structure, utility, price and cost. *IEEE Trans. on Software Eng.*, 17(4):345–356, 1991.

[2] J. Baik, N. Eickelmann, and C. Abts. Empirical software simulation for COTS glue code development and integration. In *Proc. of the 2001 Int'l Conf. on Computer Software and Applications (COMPSAC 2001)*, pages 297–302, Oct. 2001.

[3] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future lifecycle processes: COCOMO 2.0. *Annals of Software Eng.*, 1:57–94, 1995.

[4] P. Brereton and D. Budgen. Component-based systems: A classification of issues. *IEEE Computer*, 33(11):54–62, Nov. 2000.

[5] L. C. Briand, J. Wüst, S. V. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: An industrial case study. In *ICSE99*, pages 345–354, 1999.

[6] L. Brownsword, T. Oberndorf, and C. A. Sledge. Developing new processes for COTS-based systems. *IEEE Software*, 17(4):48–55, Jul. 2000.

[7] R. Burnett. A trade-off method between cost and reliability. In *Proc. of the 17th Int'l Conf. Chilean Computer Science Society (SCCC '97)*, 1997.

[8] T. Chàvez. A decision-analytic stopping rule for validation of commercial software systems. *IEEE Trans. on Software Eng.*, 26(9):907–918, Sept. 2000.

[9] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. on Software Eng.*, 20(6):476–493, Jun. 1994.

[10] K. Deb. *Multi-Objective optimization using evolutionary algorithms*. John Wiley & Sons, Chichester, England, edition, 2002.

[11] H. Erdogmus. Building a business case for COTS-centric development: An investment analysis perspective. In *Proc. of the ICSE 99 Workshop on Ensuring Successful COTS Development*, May 1999. Position Paper.

[12] M. E. Helander, M. Zhao, and N. Ohlsson. Planning models for software reliability and cost. *IEEE Trans. on Software Eng.*, 24(6):420–434, Jun. 1998.

[13] L. L. Jilani and A. Mili. Estimating COTS integration: An analytical approach. In *Proc. of the 5th Maghrebian Conf. on Software Eng. and Artificial Intelligence*, Dec. 1998.

[14] H. Jung, C. Chung, and K. O. Lee. Selecting optimal COTS products considering cost and failure rate. In *Proc. of the 10th Int'l Symp. on Software Reliability Eng. (ISSRE'99)*, 1999.

[15] J. Kontio. A case study in applying a systematic method for COTS selection. In *Proc. of the 18th Int'l Conf. on Software Eng. (ICSE'96)*, pages 201–209, Mar. 1996.

[16] S. Krishnamurthy and A. Mathur. On the estimation of reliability of a software system using reliabilities of its components. In *Proc. of the 8th Int'l Symp. on Software Reliability Eng. (ISSRE '97)*, 1997.

[17] M. Lin and X. Yongsen. An adaptive dependability model of component-based software. *SIGSOFT Softw. Eng. Notes*, 28(2):10, 2003.

[18] S. Merad and R. de Lemos. A game theoretic solution for the optimal selection of software components. Technical Report CS-TR-683, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, Sept. 1999.

[19] A. Mili, S. F. Chmiel, R. Gottomukkala, and L.Zhang. An integrated cost model for software reuse. In *Proc. of the 22nd Int'l Conf. on Software Eng. (ICSE '00)*, June 2000.

[20] M. Morisio, C. B. Seaman, A. T. Parra, V. R. B. S. E. Kraft, and S. E. Condon. Investigating and improving a COTS-based software development. In *Proc. of the 22nd Int'l Conf. on Software Eng.*, pages 32–41, Jun. 2000.

[21] J. D. Musa. *Software reliability engineering*. McGraw-Hill, New York, 1999.

[22] R. A. Paul, T. Kunii, Y. Shinagawa, and M. F. Khan. Software metrics knowledge and databases for project management. *IEEE Trans. on Knowledge and Data Eng.*, 11(1):255–264, Jan./Feb. 1999.

[23] J. Rakowska, R. T. Haftka, and L. T. Watson. Tracing the efficient curve for multi-objective control-structure optimization. *Computing Systems in Eng.*, 2(6):461–471, 1991.

[24] S. Sedigh-Ali, A. Ghafoor, and R. A. Paul. A metrics-guided framework for cost and quality management of component-based software. In A. Cechich, editor, *Component-Based Software Quality: Methods and Techniques*, volume 2693 of *Lecture Notes in Computer Science*, pages 384–414. Springer-Verlag, Germany, July 2003.

[25] S. Sedigh-Ali and A. Ghafoor. Optimization of component-based software development: an evolutionary approach. *to be submitted to IEEE Transactions on Software Engineering*, Dec. 2003.

[26] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A Bayesian approach to reliability prediction and assessment of component based systems. In *Proc. of the 12th Int'l Symp. on Software Reliability Eng. (ISSRE '01)*, pages 12–21, Nov. 2001.

[27] R. B. Statnikov and J. B. Matusov. *Multicriteria Optimization and Engineering*. Chapman and Hall, New York, edition, 1995.

[28] P. Vitharana. Risks and challenges of component-based software development. *Comm. of the ACM*, 46(8):67–72, 2003.

[29] J. M. Voas. Certifying off-the-shelf software components. *IEEE Computer*, 31(6):53–59, Jun. 1998.

[30] M. Xie. On the determination of optimum software release time. In *Proc. of the Int'l Symp. on Software Reliability Eng.*, pages 218–224, May 1991.