Evaluating Software Reuse Alternatives: A Model and Its Application to an Industrial Case Study

Amir Tomer, *Member*, *IEEE*, Leah Goldin, *Senior Member*, *IEEE*, Tsvi Kuflik, *Member*, *IEEE*, Esther Kimchi, and Stephen R. Schach, *Member*, *IEEE Computer Society*

Abstract—We propose a model that enables software developers to systematically evaluate and compare all possible alternative reuse scenarios. The model supports the clear identification of the basic operations involved and associates a cost component with each basic operation in a focused and precise way. The model is a practical tool that assists developers to weigh and evaluate different reuse scenarios, based on accumulated organizational data, and then to decide which option to select in a given situation. The model is currently being used at six different companies for cost-benefit analysis of alternative reuse scenarios; we give a case study that illustrates how it has been used in practice.

Index Terms—Reuse models, cost estimation, maintenance management, software libraries, process metrics, process measurement, planning.

1 Introduction

Software reuse is a major component of many software productivity improvement efforts because reuse can result in higher quality software at a lower cost and delivered within a shorter time [1]. One of the critical reuse success factors is the adoption of the product-line approach of Boehm [2]. This approach is detailed in [3], which describes a product line organized to operate in two circles: core-asset development and product development. Core assets are software artifacts that are developed or acquired (during operation of the first circle), are owned by the product line, and are available for acquisition as building blocks for specific products. However, it is at the discretion of product developers, performing in the other circle, to decide whether to acquire (*buy*) core assets from the product-line repository or develop their software artifacts from scratch (*make*). This decision is critical, not only for each specific product, but also for the entire product line because the central activity of core-asset development is driven by the needs of specific products, both present and future. In order to evaluate the alternatives correctly, these make/buy decisions should be based on data collected from past reuse and development efforts and on estimation of future activities. On the other hand, it is not trivial to measure all reuse activities precisely because reuse can be performed both centrally and individually for specific products. Moreover, artifacts can be transferred from one product to another, as part of opportunistic reuse efforts, and control is lost over both the configuration and the cost.

In this paper, we propose a model by which reuse activities in product lines may be systematically measured or estimated and alternative reuse scenarios may be evaluated and compared for effective support of the make/buy process. In Section 2, we give background material on software reuse and software reuse cost models. In Section 3, we describe and demonstrate, in a case study, our conceptual model of reuse operations and scenarios. In Section 4, we add cost elements to the model and address typical reuse scenarios and their costs. In Section 5, we describe how the model is being used in practice and present an industrial case study. Section 6 presents conclusions and future work.

- *A. Tomer is with RAFAEL Ltd., PO Box 2250/1P, Haifa 31021, Israel. E-mail: tomera@rafael.co.il.*
- *L. Goldin is with Golden Solutions, PO Box 6017, Kfar Saba 44641, Israel. E-mail: l_goldin@computer.org.*
- *T. Kuflik is with the Department of Management Information Systems, University of Haifa, Mount Carmel, Haifa 31905, Israel. E-mail: tsvikak@mis.hevra.haifa.ac.il.*
- *E. Kimchi can be reached at 46 Ben Gurion St. Ramat Hasharon 47321, Israel. E-mail: estikim@zahav.net.il.*
- *S.R. Schach is with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Station B Box 351679, Nashville, TN 37235. E-mail: srs@vuse.vanderbilt.edu.*

Manuscript received 10 Feb. 2004; revised 14 May 2004; accepted 2 July 2004. Recommended for acceptance by A. Mili.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0022-0204.

2 Background

2.1 Reuse Concepts

Reuse takes place when an existing artifact is utilized to facilitate the development or maintenance of the *target product*. The *scope of reuse* can vary from the narrowest possible range, namely, from one product version to another, to a wider range such as between two different products within the same line of products or even between products in different product lines. The scope of reuse is limited, in general, by the nature of and constraints on a product line; for example, it is unwise to reuse a desktop application in a mission-critical system. The *granularity of reuse* can vary from the narrowest range of reusing a single

artifact, such as a component, document, or test case, to the widest possible range, namely, a whole product. Reuse can take place during any phase of the life cycle, including marketing and proposals, requirements, analysis, design, coding, and testing. Performing reuse at a certain *level of reuse* (life-cycle phase) usually carries with it reuse at all subsequent levels. In *black-box reuse*, the artifact is reused unchanged, whereas, in *white-box reuse*, the artifact is modified to fit the target product.

In order to be able to apply reuse systematically, there is a need to study and analyze the domain in which software reuse will take place in order to identify software artifacts that are candidates for future reuse. This initial task is called *domain analysis* [4], a process of systematic studying, modeling, and analysis of the domain. Domain analysis starts with domain information gathering and yields a domain model, architecture, and list of artifacts (existing and future) that are candidates for future reuse.

2.2 Reuse Costs and Costing Policies

In many ways, reuse-based software development is similar to the usual software development process that is performed by each software organization in terms of its own software life-cycle model. Costing all the life-cycle activities involved in the development of new software artifacts for a product is therefore done in terms of the life-cycle costing policy of that organization. However, incorporating reuse into the process involves the use of artifacts and an organizational infrastructure that have been developed and established out of the context of the current product.

In the following, we will list the main costs involved in reuse-based development, as well as various policies that may be employed for the costing of the entire product development effort. Most organizations are aware of these individual costs. However, some companies have difficulty in integrating these costs into a precise company-wide costing policy. Moreover, even when a company has such a policy in place, such a costing policy is company-specific. This was our experience, for example, at the beginning of the ISWRIC project (described in detail in Section 5.1), when the seven companies involved were unable to share and compare reuse cost data because each company computed its costs in its own way. This was the main motivation for developing the model introduced in this paper.

The issue of reuse-based development costs and costing (or funding) policies is detailed in [5]. Although we take a similar general approach, we have refined the cost categorization in order to highlight the costing difficulties, which are resolved by our model.

The overall costs of reuse-based software development, from the organizational or product-line point of view, may be divided into three categories: product construction, core-asset construction, and infrastructure. The following are the costs included in each category.

2.2.1 Product Construction Costs

- **Asset acquisition cost** includes the cost of purchasing the asset from a source outside the product, as well as the effort invested in seeking the appropriate asset, whether it is stored in a core-asset repository or is available elsewhere in the organization, in the public domain, or the market.
- **Asset development cost** includes the cost of the analysis, design, and coding of new or modified artifacts, as well as the cost incurred by all the verification and validation activities performed directly on the asset, such as design reviews, code walkthroughs, and unit tests. When existing artifacts are reused as white boxes, the relevant development cost is only the cost of modifications made to those artifacts. However, in all instances of reuse, including black box and COTS, extra effort may have to be invested in modifying *other* artifacts of the product in order to be able to integrate the reused asset into that product.
- **Product integration, verification, and validation cost** includes the cost of partial and full integrations, design reviews, subsystem and system testing, and acceptance testing. It also includes the effort put in the development of the testing environment, such as test equipment, simulators, automated test procedures, etc.

2.2.2 Core-Asset Construction Costs

- **Asset acquisition cost** is the same type of cost as for product assets.
- **Asset development cost** is similar to the development cost of product asset development. Although the repository is not expected to produce complete products, and therefore the product integration, verification, and validation cost does not apply, it might produce compound assets, which may be constructed in a similar way to a complete product. In this case, the cost of integration, verification, and validation incurred until the core asset is ready for use may be considered entirely as this asset's development cost.

2.2.3 Infrastructure Costs

- **Repository establishment and maintenance cost** includes the cost of database analysis and design, tool development or purchase, database administration, etc.
- **Repository storage and cataloging cost** includes the cost of the effort needed for approving the artifacts for the repository and determining the metadata required in order to enable efficient search and retrieval of core assets in the catalog.
- **Domain Analysis (DA) cost** is incurred by a set of activities needed to define the scope and the contents of candidate reuse assets, together with locating them in past products and making them available for the core-asset repository. One important activity in DA is to define the categories into which core assets are classified in order to check commonality among candidate reusable artifacts. These categories are later incorporated into the catalog metadata accompanying the assets in the repository. Another DA activity is the "mining" of candidate assets from

past products, in terms of the defined scope and categorization of artifacts within the repository. DA is usually performed when a product line is initiated and established. However, because DA may be reapplied whenever the scope of the product-line changes (for example, when entering a new business niche), it may be considered an on-going activity, with accumulated costs.

Because the core asset construction costs and the infrastructure costs previously mentioned are incurred centrally, at the organization (or the product-line) level, a costing policy is needed that defines the way these costs are distributed among target products. A number of examples of such costing policies ("funding strategies") are found in [5]. However, these strategies are explained in words, providing no practical tool for calculating the costs precisely.

The cost model introduced in this paper provides a systematic and straightforward way of calculating the overall cost of various reuse alternatives in order to select the one that is the most cost-effective. When our model is described later, we will refer to the costs mentioned in this section and show how they should be considered in terms of a specific costing policy.

2.3 Related Work

Software reuse is not merely a technical issue. On the contrary, it is widely accepted that the organizational challenges of software reuse outweigh the technical ones, as summarized by the "STARS" program report, for example [6]. As a result, metrics are needed in order to make "business decisions possible by quantifying and justifying the investment necessary to make reuse happen" [7]. The importance of economic models and metrics of software reuse is widely recognized, as can be seen from numerous existing models, such as the measure put forward by Barnes and Bollinger [8], who suggested analytical approaches to making good reuse investments, based on cost-benefit analysis. Another example is the cost-benefit analysis with net present value model of Malan and Wentzel [9]. There are many more models, as surveyed by Wiles [10], Poulin [7], Lim [11], and others.

Various economic models can be used to show that successful software reuse is possible in theory. In fact, there are many software reuse success stories in which the economic benefits are evident in practice. For example, substantial costs were saved due to implementation of software reuse in the "STARS" demonstration project. The first system that was developed cost 43 percent of a reference baseline and a second cost only 10 percent [6]. Another example is the experience at Hewlett-Packard described in [11] where, by applying software reuse, a defect reduction of 15 percent was achieved and productivity increased by 57 percent. Several additional reuse success stories are presented by Poulin [7].

Organizations are usually in various stages of "the incremental adoption of reuse" described by Jacobson et al. [1] as a practical approach for transition toward becoming a domain-specific reuse-driven organization. However, the transition phase is lengthy, and every organization needs to find its own way; it is not obvious that the same approach is good for everyone. Hence, organizations are at different stages of the transition phase, and project managers must decide what course to take at every point in time, based on their analysis of the current situation. None of the previously mentioned models and evaluations provides a means to analyze and weigh alternative approaches for software development with various types of reuse (including no reuse). In most instances, the point of reference of these models is software development without any reuse at all. Even when reuse is an alternative, it is common to find examples of only white-box reuse.

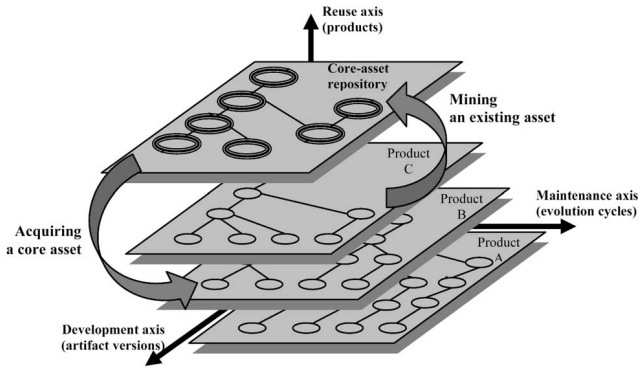

Fig. 1. The three-dimensional evolution of a software product line.

What is needed, in addition to a detailed economic model, is a framework for analysis and comparison of development approaches: without reuse, with white-box reuse, and with black-box reuse. We believe that the economic measurements used to assess the benefits of the entire software reuse effort can also be used to help developers in the "make versus buy" assessment. A systematic definition of various reuse scenarios that will provide a means to weigh the different alternatives, based on past experience and estimation, is now needed.

3 The Underlying Model

The reuse cost model described in this paper is based on a three-dimensional model, introduced in [12], in which all the activities of software construction in product lines, as well as the evolution of all the artifacts involved, are described along three axes: development, maintenance, and reuse (see Fig. 1). A key point in the underlying three-dimensional model and in the product-line approach in general is the existence of a core-asset repository and the discipline that artifacts cannot be freely transferred between specific products without first being stored and cataloged in the core-asset repository. The activity of fetching reuse candidates from specific products and copying them into the repository is called *mining*, whereas the inverse operation of copying artifacts from the repository into specific products, in order to reuse them, is called *acquisition* [3]. For simplicity, the model described in this paper is only two-dimensional; specifically, we distinguish only between the reuse operation, that is, copying artifacts from one plane to another along the reuse axis, and all other operations along the development and maintenance axes that are

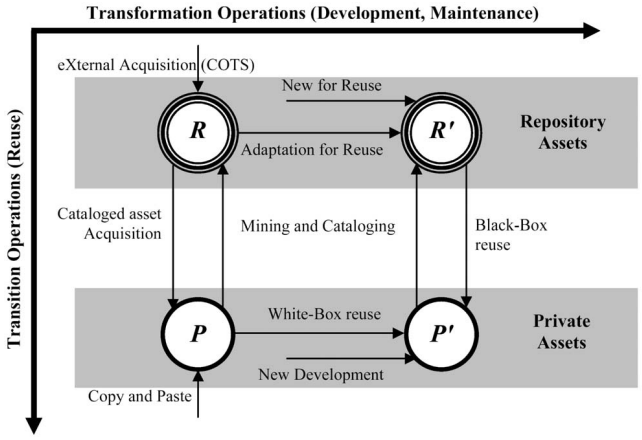

Fig. 2. Assets and reuse operations.

concerned with making modifications to artifacts within the same plane. In other words, for simplicity, we have unified the development and maintenance axes, as will be described.

3.1 Asset Types

We assume that there is a core-assets repository in which artifacts and their relevant metadata are indexed. There is no need, however, for the artifacts themselves to be physically stored in the repository—they may reside in any specific product library, provided that each artifact preserves its unique identification, namely, its type and the identification of the specific product to which it belongs [13].

We define two kinds of artifacts, as follows:

- **Repository Assets**: Artifacts that are cataloged in the core-asset repository, including their metadata. These artifacts are available either for acquisition by specific products or for modification (for further reuse purposes) within the core-asset repository itself. One important attribute of an artifact is its size or some other measure by which the complexity of two different assets may be compared.
- **Private Assets**: Artifacts that are contained within a specific product and are available either for mining and cataloging or for private modification, but only within the environment of the same specific product.

Representatives of the two asset types appear in Fig. 2. The thick perimeter of repository assets denotes that they are "wrapped" in metadata (catalog information).

Next, we define a number of reuse-related operations, which typically are performed during software development.

3.2 Elementary Operations

In terms of the underlying three-dimensional model, we identify an elementary operation as an activity for obtaining an asset, *performed along a single axis*. However, in contrast to the usual type of software product construction, the scope of reuse-based software construction extends beyond a single product. Based on the assumption that each product development is performed within its own budget, this extension is significant with respect to cost. We therefore unified the development and maintenance operations of the original model into one category of elementary operations, denoted as *Transformation Operations* (along the horizontal axis), whereas reuse operations (along the vertical axis) will be denoted here as *Transition Operations* (see Fig. 2). Each of the operations described here may be performed independently of other operations. Later, we will define sequences of operations as *scenarios*.

3.2.1 Transformation Operations

Adaptation for Reuse (*AR*): Modifying an existing repository asset R, resulting in another reusable repository asset R'. Note that both R and R' must reside in the repository.

New for Reuse (*NR*): Constructing a new repository asset R' from scratch. It is expected that the asset will be developed in conformance with applicable standards that make it effectively reusable.

White-Box reuse (*WB*): Modifying an existing private asset P into another private asset P' within the same product. Both P and P' must reside in the product library as revisions of the same artifact.

New Development (*ND*): Constructing a new private asset P' from scratch.

3.2.2 Transition Operations

Cataloged asset Acquisition (*CA*): Acquiring a copy of a repository asset R for a specific product as a private asset P. It is assumed that P needs to undergo further modifications (white-box reuse) within the product in contrast to black-box reuse (see next reuse operation).

Black-Box reuse (*BB*): Acquiring a copy of a repository asset R' "as is" (that is, with no modifications) for a specific product as a private asset P'. Ideally, this should be an elementary copy operation; in practice, however, this operation may require some overhead activities as a consequence of adapting the architecture of the target product in order for the imported asset to fit. This is also the case when acquiring COTS assets for the product.

Mining and Cataloging (*MC*): Identifying and acquiring an existing private asset P, from a certain product, and then storing and cataloging it formally as a repository asset R.

Copy and Paste (*CP*): Acquiring a copy of a private asset P for a specific product. The source asset is not cataloged in the repository, and awareness of its existence is based on personal knowledge.

eXternal Acquisition (*XA*): Acquiring an asset from some external source and cataloging it as a repository asset R. This is the case with COTS artifacts.

3.3 Reuse Scenarios

We define any sequence of elementary operations as a *scenario*. As an example for such a scenario, we start this section with a short case study, which will be utilized in the sections that follow.

Case Study 1. Two software products are being developed simultaneously, each of which needs to employ a queue of elements. Although one of the products is to be programmed in Java and the other in C++, the software manager realizes that both need a queue class design, which will apparently be identical in both products. Moreover, a previous product has implemented a queue package in Ada, which provides the same functionality. Also, anticipating

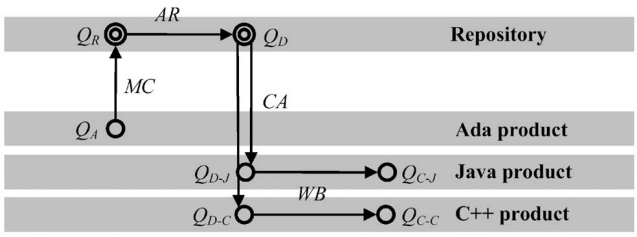

Fig. 3. Reuse scenario of Case Study 1.

that a queue class may be useful for further products in the future, the software manager decides to assign a software engineer to reverse-engineer the Ada package and turn it into an object-oriented queue class design. The entire effort may now be decomposed in terms of the elementary reuse operations, as follows (see Fig. 3):

1. First, the Ada queue package (Q_A) is identified in the source product, copied into the repository, and complemented with metadata, resulting in a repository queue package (Q_R). This operation was previously defined as Mining and Cataloging (*MC*).
2. Because there was no reusable design available, the repository queue package is reverse-engineered, obtaining a queue class design in the repository (Q_D), ready for reuse by both products. This operation was previously defined as Adaptation for Reuse (*AR*).
3. Next, a copy of the queue class design is obtained by both the Java product (Q_{D-J}) and the C++ product (Q_{D-C}). These operations have been defined as Catalog Acquisition (*CA*).
4. Finally, the products need the class design to be turned into code classes in Java (Q_{C-J}) and C++ (Q_{C-C}), respectively. This is considered to be White-Box reuse (*WB*) because private modifications were required for both products.

The sequence of operations performed, namely, $MC \rightarrow AR \rightarrow CA \rightarrow WB$, is the *scenario* the software engineers performed in order to obtain the two queue classes from the Ada queue package. Later, we will compare the cost of this scenario with the cost of other possible scenarios for achieving the same result.

A *reuse scenario* is any sequence of elementary operations performed while practicing reuse. In order to determine the optimal scenario through which the final target can be obtained from the original source assets, we must be able to compare the relative cost of alternative scenarios. Accordingly, next we define the cost of the elementary operations from which the cost of typical reuse scenarios can be computed or estimated and compared.

4 The Cost of Reuse

4.1 Elementary Cost Components

We can now associate a cost factor with each of the nine basic reuse operations of Section 3.2. In general, each operation involves two assets: a source asset, the asset to be copied or modified, and a target asset, the copy or the modification obtained from the source. New assets, though, are obtained from scratch. However, in order to deal uniformly with all cases, we will assume the existence of a null asset, denoted by $\emptyset$, which is considered to be the source asset in cases in which the target asset is new. This is reflected in Fig. 4.

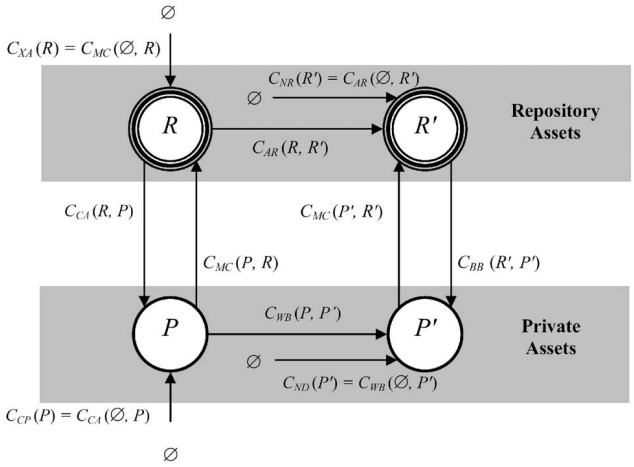

Fig. 4. The elementary components of the cost of reuse.

4.1.1 Cost of Transformation Operations

$C_{AR}(R, R')$ is the cost of Adaptation for Reuse (*AR*), derived from the direct effort (in person-hours, for example) invested in obtaining repository asset R' from another repository asset R, in a way that will result in the compatibility of R' with the reuse standards applicable for core assets in the product line.

$C_{NR}(R') = C_{AR}(\emptyset, R')$ is the cost of developing R' from scratch as New for Reuse (*NR*).

$C_{WB}(P, P')$ is the cost of White-Box reuse (*WB*), derived from the direct effort invested in performing private modifications to P, at the specific product level, to obtain P'. In practice, the cost of white-box reuse is usually smaller than adaptation for reuse. Private modification focuses on only those changes needed to adapt the private asset for its specific needs, within the context of the specific product, whereas adaptation for reuse takes into consideration several products that might reuse that asset.

$C_{ND}(P') = C_{WB}(\emptyset, P')$ is the cost of constructing the target private artifact P' from scratch as New Development (*ND*).

4.1.2 Cost of Transition Operations

$C_{CA}(R, P)$ is the cost of Catalog Acquisition (CA), which is the cost involved in replicating a copy of the repository asset R as a private asset P; this cost also includes the search and evaluation efforts before an asset is selected. Although in most cases this cost is expected to be close to zero, license or royalty fees may apply as well.

$C_{CP}(P) = C_{CA}(\emptyset, P)$ is the cost of Copy and Paste (CP), incurred in searching for and acquiring an uncataloged artifact from some external source.

$C_{MC}(P, R)$ is the cost of Mining (M) a copy of a private asset P from a specific product and Cataloging (C) it as a repository asset R. This cost typically includes determining

TABLE 1
A Costing Policy

Operations \ Costs		Product construction			Core-asset construction		Infrastructure		
		Asset acquisition	Asset development	Product integration, verification and validation	Asset acquisition	Asset development	Storage and cataloging	Repository establishment and maintenance	Domain analysis
Transformation	$C_{AR}(R, R')$					Full	Full		
	$C_{NR}(R')$					Full	Full		
	$C_{WB}(P, P')$		Full	Full [2]					
	$C_{ND}(P')$		Full	Full [2]					
Transition	$C_{CA}(R, P)$	Full						Part [3]	
	$C_{CP}(P)$	Full							
	$C_{MC}(P, R)$				Full		Full	Part [3]	Part [4]
	$C_{XA}(R)$				Full		Full	Part [3]	
	$C_{BB}(R', P')$	Full	Full [1]	Full [2]				Part [3]	

Notes:

[1] "Full" here refers to the cost incurred in modifying other artifacts, in order to enable the integration of the black-box asset.

[2] "Full" here means that the full cost of integrating the specific asset into the complete product is considered, as well as the cost of asset verification and validation within the context of the complete product. This applies to the operations that result in assets for IV&V, namely, new development, white-box reuse, and black-box reuse.

[3] The total cost of repository establishment and maintenance may be "charged" in part to every transition operation accessing the repository. In the case studies reported in this paper, we chose this value to be "free of charge," that is, 0.

[4] The main outcome of a domain analysis is a set of artifacts found in past products, which need to be mined and cataloged into the repository, as candidates for core assets (in their original form or after adaptation for reuse). Therefore, the total cost of the domain analysis may be distributed among the artifacts of this set. In the case studies reported in this paper, we decided that the domain analysis cost would be divided over the mining and cataloging operation of the reused artifacts, in direct proportion to their size/complexity.

R's metadata and storing the metadata in the repository for reuse. MC does *not* include any changes made to the artifact itself.

$C_{XA}(R) = C_{MC}(\emptyset, R)$ is the cost of eXternally Acquiring (XA) a COTS asset R and cataloging it in the repository. This is usually the purchase price.

$C_{BB}(R', P')$ is the cost of Black-Box (BB) reuse. It is the cost of the direct effort invested in acquiring a cataloged repository asset R' and integrating it into a specific product. Because the reused artifact itself is not changed, there is no modification cost as such. However, extra effort may be required to adapt the architecture, interfaces, or other artifacts of the target product in order to enable the asset to be integrated. This is a typical cost when COTS artifacts are considered for a product.

4.2 Applying a Costing Policy

In Section 2.2, we addressed the typical costs incurred during reuse-based development (reuse costs). In the following, we show how costing policies are used to incorporate these costs in our model. We assume that the reuse costs can be measured or estimated by the company, but the difficult question is how to apply these costs to specific reuse scenarios.

We view a costing policy as a mapping between the reuse costs of Section 2.2 and the costs of the reuse-related operations of Section 4.1. Table 1 shows an example of such a policy. Using this table, the cost of each basic operation is the sum of the applicable reuse costs, along the relevant row, as follows:

- An empty entry denotes that the corresponding cost is not applicable to the operation in question and, therefore, its contribution to the total is zero. For example, asset acquisition cost is not applicable to any transformation operations because they are performed only on artifacts that already exist in the product or in the repository.
- "Full" means that the measured or estimated cost is charged in full to the cost of the operation. This is usually the case when all of the cost is incurred in

Fig. 5. Pure development scenario.

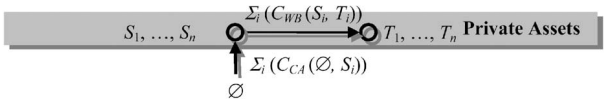

Fig. 6. Opportunistic reuse scenario.

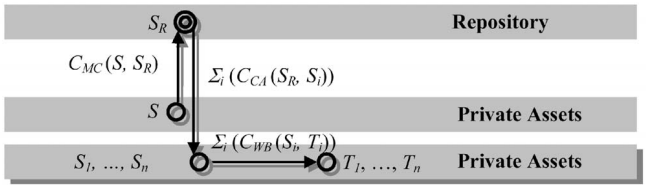

Fig. 7. Controlled reuse scenario.

obtaining the resulting artifact. For example, the full product development cost is charged to the white-box reuse ($C_{WB}(P, P')$) operation, reflecting the direct cost of obtaining P' from P.

- "Part" means that part of the measured or estimated cost is charged to the cost of the operation. This is usually the case when costs are incurred during activities that do not necessary involve the resulting artifact. For example, the cost of repository establishment and maintenance is partially charged to any transition operation accessing the repository.

We remark that the concept of the costing policy table can be easily adapted to any other cost directives affecting the process (training, for example) by adding a new column to the table and deciding how to allocate the total cost to the basic reuse operations. We also believe that this tool can be easily employed in formalizing the various funding strategies of [5].

4.3 Typical Reuse Scenarios and Their Costs

In Case Study 1, we demonstrated a specific reuse scenario, whose total cost, according to the costing policy of Table 1, is

$$C_{MC}(Q_A, Q_R) + C_{AR}(Q_R, Q_D) + C_{CA}(Q_D, Q_{D-J}) + C_{CA}(Q_D, Q_{D-C}) + C_{WB}(Q_{D-J}, Q_{C-J}) + C_{WB}(Q_{D-C}, Q_{C-C}),$$

where

1. $C_{MC}(Q_A, Q_R)$ is the cost of Mining and Cataloging Q_A into Q_R,
2. $C_{AR}(Q_R, Q_D)$ is the cost of reverse-engineering the queue class design Q_D,
3. $C_{CA}(Q_D, Q_{D-J})$ and $C_{CA}(Q_D, Q_{D-C})$ are the costs of acquiring Q_D for the specific products, and
4. $C_{WB}(Q_{D-J}, Q_{C-J})$ and $C_{WB}(Q_{D-C}, Q_{C-C})$ are the costs of programming the Java and the C++ queue classes, respectively.

The total cost can easily be compared to the cost of developing both Q_{C-J} and Q_{C-C} from scratch for the purpose of estimating the cost-effectiveness of reusing Q_A in this fashion.

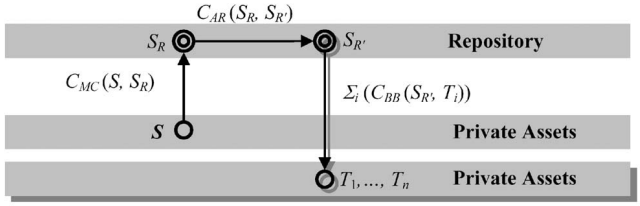

Fig. 8. Systematic reuse scenario.

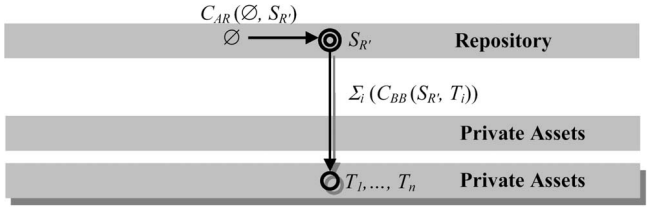

Fig. 9. Alternative systematic reuse scenario.

We now generalize the case study and consider a variety of typical reuse scenarios. In each of these scenarios, there are n products in which product i requires target software component $T_i, i = 1, \ldots, n$. We further assume that there is significant commonality among the $T_i, i = 1, \ldots, n$. In addition, there exists a private component S in an existing product from which every T_i may be obtained by performing certain modifications. We consider the following typical scenarios, from pure development to systematic reuse:

Pure Development (*PD*) *scenario*. This is the case in which each group responsible for one of the n products is unaware of the existence of S and, therefore, develops its target component T_i from scratch (see Fig. 5). In Figs. 5, 6, 7, 8, and 9, shadowed arrows represent repeated application of the same activity in separate products.

Assuming that the cost of developing a new component T_i is $C_{ND}(T_i) = C_{WB}(\emptyset, T_i)$, the total cost of the PD scenario is

$$\Sigma_i(C_{WB}(\emptyset, T_i)).$$

Opportunistic Reuse (*OR*) *scenario*. In this case, each group responsible for one of the n products knows that there exists a viable source S, but it is not stored in any shared repository or registered in a public catalog. Therefore, each of the products invests independently in searching for S, obtaining a copy of it as a private source asset S_i (at a cost of $C_{CP}(S_i) = C_{CA}(\emptyset, S_i)$) and then modifying it (white-box reuse) for utilization in target component T_i (see Fig. 6).

Thus, the cost of the entire OR scenario is

$$\Sigma_i(C_{CA}(\emptyset, S_i)) + \Sigma_i(C_{WB}(S_i, T_i)).$$

Although opportunistic reuse may result in some savings for a specific product, it is nevertheless performed on an individual product basis. Opportunistic reuse therefore does not benefit from common activities or a public repository.

Controlled Reuse (*CR*) *scenario*. Suppose that a core-asset repository has been established in which assets are stored or cataloged for the benefit of other products. Suppose further that a group, independent of any specific product, is responsible for looking for private assets in specific products, mining them, and cataloging them in the repository. In this

scenario, a copy of the private component S is first mined and cataloged as a repository asset S_R (at the cost of $C_{MC}(S, S_R)$). Other than the mining and cataloging effort, no other effort is invested in modifying S, and it is made available for other products for acquisition in its original form. Thus, each of the products will acquire a copy S_i of S_R (at a cost of $C_{CA}(S_R, S_i)$) and then reuse S_i (white-box reuse), similarly to the previous scenario. The total cost of the CR scenario is then (see Fig. 7)

$$C_{MC}(S, S_R) + \Sigma_i(C_{CA}(S_R, S_i)) + \Sigma_i(C_{WB}(S_i, T_i)).$$

At first sight, the savings of the CR scenario over the OR scenario do not appear to be significant. However, in practice, the mining and cataloging effort is performed just once in the CR scenario, in contrast to n times in the OR scenario. Furthermore, it is likely that cost of the catalog acquisition is close to zero.

Systematic Reuse (*SR*) *scenario*. The ultimate goal of reuse processes is to have a set of assets that are readily available for reuse in all future products without further modification. In our case, a copy of S will first be mined and cataloged as a repository asset S_R, at the cost of $C_{MC}(S, S_R)$. Then, S_R will undergo modifications to convert it into a reusable assets $S_{R'}$, at the cost of $C_{AR}(S_R, S_{R'})$. Finally, each of the products will acquire a copy of $S_{R'}$ as its private target asset T_i, (black-box reuse). The cost of integrating $S_{R'}$ into product i is $C_{BB}(S_{R'}, T_i), i = 1, \ldots, n$. The total cost of the S_R scenario is therefore (see Fig. 8)

$$C_{MC}(S, S_R) + C_{AR}(S_R, S_{R'}) + \Sigma_i(C_{BB}(S_{R'}, T_i)).$$

An alternative form of systematic reuse is based on New for Reuse development (instead of adaptation). In this case, shown in Fig. 9, the cost would be

$$C_{NR}(\emptyset, S_{R'}) + \Sigma_i(C_{BB}(S_{R'}, T_i)).$$

5 Industrial Experience

5.1 Implementation by the Software Industry

A version of this model was implemented and deployed by ISWRIC (Israel SoftWare Reuse Industrial Consortium), a joint project of seven leading Israeli industrial companies supported, in part, by the Chief Scientist of the Israeli Ministry of Trade and Industry. Three of these companies are mainly defense contractors, developing specific customer-oriented systems, whereas the other four produce off-the-shelf or customizable commercial products. The model is deployed as a common measurement tool with which data from industrial pilot projects are collected and analyzed.

This two-phase, three-year project commenced in June 2000. During the first phase, a common software reuse methodology was developed, based on existing approaches and practices, but modified to take into account the specific requirements of the consortium members. During the second phase, all seven participating companies implemented the methodology in real projects. Each company tailored the model to the specific needs of its pilot projects and evaluated the methodological aspects relevant to the pilot projects and the company.

The model is currently being used to compute and compare the potential costs and benefits of alternative reuse scenarios. It is also used for periodical reports of economic aspects of software reuse programs. Of the seven participating companies, six have defined and evaluated their reuse scenarios using the model and have used it as an aid to justify their selected reuse approach. Scenarios defined by these six companies include:

- Systematic reuse, implemented by five companies. Of these companies, three are implementing systematic reuse based on new development of reusable assets, and two are implementing systematic reuse based on existing assets adapted for reuse.
- Controlled reuse, implemented by one company. This company has selected a reuse policy in which the reusable assets are centrally brought to a predefined degree of maturity from where each product can adapt these assets (by means of white-box reuse) to its specific needs.

The data were gathered, presented, and analyzed periodically by the consortium members using the model, as part of periodic project reports and information exchange.

Use of the model in the industry to date has demonstrated the potential benefits of the various reuse scenarios. The model also provided a common ground for the presentation and discussion of results at ISWRIC meetings.

5.2 Description of an Industrial Case Study

In this section, we show how the cost model was used at Tadiran Electronic Systems Ltd. (TES), one of the seven companies participating in ISWRIC. The results are based on data collected at TES from 2000 to 2003.

TES is a systems house that develops large-scale software-intensive systems to meet specific requirements as defined by its customers. Most projects have a well-defined time scale and a budget under a strict contract. Products are divided into "families" of significant similarity. During the course of time, the company has acquired many software assets that are potentially reusable. In practice, however, the only reuse performed has been opportunistic, based on individual knowledge. In view of the uncertainty of the market, it is difficult to anticipate the potential number of future reuses of an asset and, therefore, the company faces a constant conflict between the immediate needs of products versus the need to invest in a long-term infrastructure.

As part of TES's participation in ISWRIC, TES decided to adopt the Controlled Reuse scenario for the most part, but the Systematic Reuse scenario was followed in several specific cases. The company established a reuse repository in which assets were cataloged along with their metadata (but not necessarily ready for black-box reuse). It was decided to allocate resources to each product team in order to perform the Adaptation for Reuse activity within the scope of their product, thus allowing the team to take into account the needs of all current and future anticipated reuses of the asset.

The reuse cost model was implemented in one department of TES. As a first step, experienced systems and software engineers performed a thorough domain analysis

TABLE 2
The Cost of Basic Operations for Seven Assets

Basic operation \ Asset	1	2	3	4	5	6	7
Mining and Cataloging (MC) [1]	**110**	**110**	**110**	**110**	**110**	**110**	**110**
Copy and Paste (CP) [2]	110	110	110	110	110	110	110
New for Reuse (NR)	1000	2000	1000	1000	200	2000	2000
Adaptation for Reuse (AR)	**500**	**500**	**400**	**400**	50	400	**1000**
Black-Box reuse (BB) [3]	**100**	**200**	**150**	**100**	30	400	**10**
New Development (ND)	700	1500	700	700	180	1500	1500
Catalog Acquisition (CA) [4]	100	100	50	50	**20**	**50**	50
White-Box reuse (WB) [5]	300	300	400	300	**50**	**300**	700
Number of reuses [6]	2	3	3	2	3	3	4

Notes:
[1] The total cost of Domain Analysis (770 p-h) was evenly distributed among all 7 assets.
[2] The cost of CP is estimated to be identical to MC, that is "Domain Analysis" for a single asset.
[3] This is the average cost of asset integration over all the reuses of this asset in the pilot study.
[4] This cost includes the effort invested (or that may be invested) in searching for an appropriate asset in the repository, then analyzing and evaluating its fit with the target product.
[5] This is the average cost of learning the asset's structure plus asset modification and adaptation over all the reuses of this asset in the pilot study.
[6] The number of target products that reused the asset during the pilot study.

Costs in boldface were measured, the others were estimates.

in the department. As a result of this analysis, a software architecture common to most products in the department was defined, and assets that share a high degree of commonality among products were identified as candidates for reuse. These assets were analyzed in depth and later were cataloged in a reuse repository, along with their metadata. Eventually, it was decided that some of them would be adapted for reuse (by adding functionality, enveloping, improving quality and documentation, etc.), whereas the others would be rewritten as new assets (because of their poor quality, technological changes, etc.). The adaptation for reuse was performed within the context of three large-scale projects that were running concurrently.

5.3 Results of the Industrial Case Study

The cost model was used to evaluate the cost-effectiveness of the selected reuse scenarios, over other alternative scenarios, with respect to seven assets. Four of the assets (numbered 1 through 4) were software modules that interface with special-purpose sophisticated hardware elements, whereas the other three (numbered 5 through 7) were generic human-computer interface modules that have to be configured for specific systems.

First, the cost (in person-hours) of each of the basic operations was determined, as shown in Table 2. The costs in boldface were actually measured, whereas the others were estimated by experienced software and systems engineers. The total cost of the domain analysis, 770 person-hours, was evenly distributed among the seven assets. Assets 1, 2, 3, 4, and 7 were adapted for black-box reuse and then were integrated into their target products. Assets 5 and 6 were stored in the repository in their original form, after domain analysis, and later were acquired by their target products and adapted for specific use by applying white-box reuse. The table also includes the number of products that reused each asset in the pilot study.

As previously mentioned, the seven assets went through two different scenarios (see Section 3.3): Assets 1, 2, 3, 4, and 7 went through the Systematic Reuse scenario, comprising mining and cataloging (*MC*), adaptation for reuse (*AR*), and black-box reuse (*BB*) activities. Assets 5 and 6 went through the Controlled Reuse scenario, comprised of mining and cataloging (*MC*), catalog acquisition (*CA*), and white-box reuse (*WB*). The calculation of the respective costs of these scenarios, based on the data of Table 2, was performed using the following formulas, where n represents the number of reuses of the asset in the pilot study:

$$\text{Systematic Reuse cost (adapted assets)} = C_{MC} + C_{AR} + n \times C_{BB},$$

$$\text{Controlled Reuse cost} = C_{MC} + n \times (C_{CA} + C_{WB}).$$

The costs of the actual scenarios for each of the seven assets are shown in boldface on the first and in the third line of Table 3. In order to evaluate the cost-effectiveness of these scenarios, they were compared with three other possible scenarios (see Section 3.3):

- Systematic Reuse of newly developed assets, comprised of Mining and Cataloging (*MC*) (which had been done anyway), New for Reuse (*NR*), and Black-Box reuse (*BB*);

TABLE 3
The Cost of Alternative Reuse Scenarios for Seven Assets

Scenario \ Asset	1	2	3	4	5	6	7
Systematic Reuse — adapted [1]	**810**	**1210**	**960**	**710**	250	1710	**1150**
Systematic Reuse — new [2]	1310	2710	1560	1310	400	3310	2150
Controlled Reuse	910	1310	1460	810	**320**	**1160**	3110
Opportunistic Reuse	820	1230	1530	820	480	1230	3240
Pure Development	1400	4500	2100	1400	540	4500	6000

Notes:
[1] The core assets were adapted from artifacts mined during domain analysis.
[2] The core assets were developed from scratch.

- Opportunistic Reuse, comprised of (separately for each product) copy and paste (*CP*) and White-Box reuse (*WB*); and
- Pure Development, comprised of New Development (*ND*).

The calculation of the respective costs of these scenarios, based on the data of Table 2, was performed using the following formulas, where n represents the number of reuses of the asset in the pilot study:

$$\text{Systematic Reuse cost (new assets)} = C_{MC} + C_{NR} + n \times C_{BB},$$
$$\text{Opportunistic Reuse cost} = n \times (C_{CP} + C_{WB}),$$
$$\text{Pure Development cost} = n \times C_{ND}.$$

We deduce from Table 3 that, except for Asset 5, the reuse scenario that was implemented in practice achieved the least cost over all its alternatives. As for Asset 5, Table 3 shows that the cost would have been less if Systematic Reuse (adapted) had been implemented instead of Controlled Reuse.

In addition to the costs themselves, we can now give a more quantitative meaning to cost-effectiveness. Suppose that we chose to implement reuse scenario S over alternative reuse scenario S'. Suppose also that we measured C_S, the cost of S, while estimating $C_{S'}$, the cost of S'. As a consequence of our choice, we nominally saved $C_{S'} - C_S$, which represents $(C_{S'} - C_S)/C_{S'}$ percent savings. For example, the cost of Systematic Reuse for Asset 1 was 810 p-h, whereas the alternative Pure Development scenario was estimated at 1,400 p-h. Therefore, the savings were (1,400 - 810)/1,400, or 42 percent.

Table 4 presents the cost-effectiveness of the actual reuse scenarios implemented for each of the assets relative to three other scenarios: the counterpart scenario, Opportunistic Reuse (which has been popular in the company), and Pure Development (which always appears to be the natural alternative).

We can see from Table 4 that, if the choice were between only Systematic Reuse and Controlled Reuse, then the largest relative savings (63 percent) would have resulted from the Systematic Reuse of Asset 7. Conversely, the worst choice would have been the Controlled Reuse of Asset 5 at a cost of 28 percent relative to the alternative. In comparison to the other referenced scenarios, we can see that the relative savings obtained by implementing the preferred scenario over Opportunistic Reuse were between 1 and 65 percent. In comparison to Pure Development, the relative savings were even more dramatic—between 41 and 81 percent.

TABLE 4
Relative Savings of Alternative Reuse Scenarios

Ratio \ Asset	1	2	3	4	5	6	7
Systematic Reuse vs.							
Controlled Reuse	**11%**	**8%**	**34%**	**12%**			**63%**
Opportunistic Reuse	**1%**	**2%**	**37%**	**13%**			**65%**
Pure Development	**42%**	**73%**	**54%**	**49%**			**81%**
Controlled Reuse vs.							
Systematic Reuse					**-28%**	**32%**	
Opportunistic Reuse					**33%**	**6%**	
Pure Development					**41%**	**74%**	
Number of reuses	2	3	3	2	3	3	4

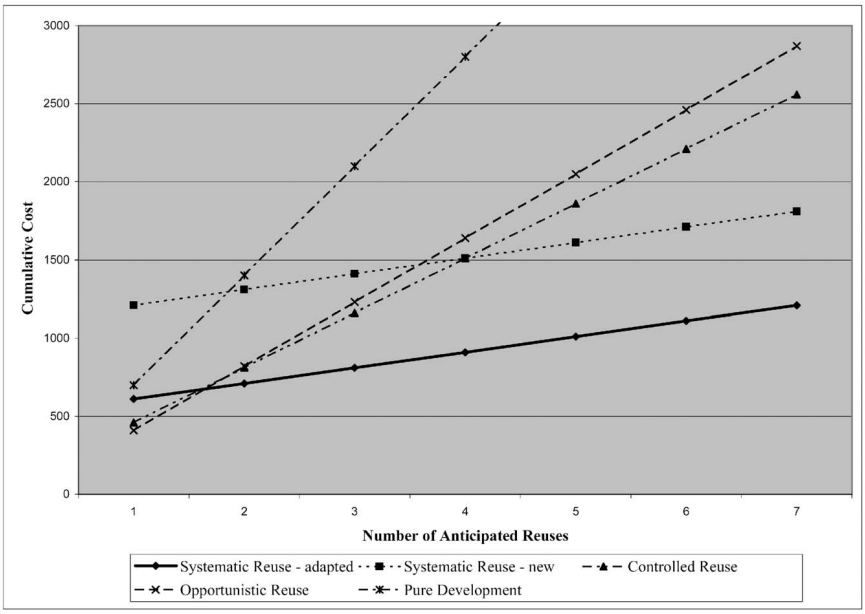

Fig. 10. Alternative reuse scenarios for Asset 4.

The data and the analysis of the industrial case study of TES validate the capability of our model to express reuse cost-effectiveness in a simple and straightforward way. Nevertheless, the most significant power of the model lies in its ability to predict the relative cost-effectiveness of future reuse, in order to select the best scenario. This prediction is calculated, as previously shown, by aggregating estimations of simple and well-defined activities. It is not surprising that the most significant parameter is the anticipated number of future reuses of the relevant asset. In general, we cannot predict this number accurately. However, calculating the costs of a number of alternative scenarios for a variable number of future reuses may yield the cost-effectiveness trend, revealing the break-even point for each of those alternatives.

Fig. 10 shows the cumulative cost of reusing Asset 4, for all five scenarios of Table 3, for anticipated number of reuses from 1 to 7. It is clear that the Systematic Reuse scenario, based on adapting existing assets, is the most beneficial from the second reuse onward. However, when only one reuse is anticipated, then Opportunistic Reuse is preferred over Controlled Reuse; Systematic Reuse can be tolerated.

6 Conclusions and Further Work

The model described in this paper is a powerful tool that can be utilized to evaluate and compare all possible scenarios of reuse-based development, regardless of the specific costing policies used. The major contribution of this model is the clear identification of the basic operations involved, together with the ability to associate a cost component with each basic operation in a focused and accurate way. The evaluation is largely based on estimated data and is therefore subject to the inaccuracies of estimation. Moreover, in different reuse contexts (such as different units in an organization, applying reuse at different times, or different individuals applying reuse), estimation might vary significantly. However, when this model is applied consistently within an organization, the accumulated data can be analyzed for better estimates in the future. We are currently analyzing the data obtained from the six companies in order to define learning curves and derive "organizational factors" that can be applied for better estimations.

There are two other contributions. First, the cost model provides a practical tool for developers to weigh and evaluate different reuse scenarios, based on accumulated organizational data, and decide what option to choose in a given situation. Second, it provides a systematic mechanism for management to analyze and evaluate various reuse alternatives at the organizational level.

As described in Section 3, this model is derived from the 3D evolution-tree model [11]. In fact, it is a projection of the entire evolution tree onto the 2D plane defined by the transition operations (reuse) axis and the transformation operations (development and maintenance) axis. However, we are currently investigating a broader model, which covers all aspects of reuse-based development.

This paper focuses on comparison of cost, which is usually the most important factor in reuse-based development. However, other factors, such as time-to-market and product quality, are also expected to improve by reusing software assets. We are currently working on extending our model to include these other factors, too.

The discussion of cost in this paper is viewed from the standpoint of the entire product line and considers the cumulative cost of the entire reuse effort. However, managers of an individual product are concerned with the direct cost of that product within the context of the appropriate infrastructure (such as a repository) and central activities (such as domain analysis). It is often expected that these will be financed centrally, by the organizational R&D group, for example. The development of a cost model,

based on that costing policy, is another direction for future research.

The model was developed within the framework of the Israeli Software Reuse Industrial Consortium, a group of seven leading Israeli systems developers. The mutual lessons learned by the consortium, the different software reuse scenarios applied by them, and the use of our model in the different scenarios are other issues for future research.

ACKNOWLEDGMENTS

The authors would like to thank the members of both the management and technical committees of the ISWRIC project who worked together and contributed numerous useful suggestions toward clarifying and stabilizing the model: Michael Vinokur (IAI—Israel Aircraft Industry), Itzhak Lavi (IAI), Varda Barzilay (ECI Telecom), Arieh Stroul (Creo), Shlomit Morad (Orbotech), Guy Pe'er (Orbotech), Rami Rashkowitz (Rafael), Anat Grynberg (NICE), and Moshe Salem (Iltam). The work of Stephen R. Schach was supported in part by the US National Science Foundation under grant number CCR-0097056.

REFERENCES

[1] I. Jacobson, M. Griss, and P. Johnsson, *Software Reuse, Architecture, Process, and Organization for Business Success.* Addison-Wesley, 1997.
[2] B. Boehm, "Managing Software Productivity and Reuse," *Computer,* vol. 32, no. 9, pp. 111-113, Sept. 1999.
[3] P. Clements and L.M. Northrop, *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2001.
[4] *Domain Analysis and Software Systems Modeling,* R. Prieto-Díaz, and G. Arango, eds. IEEE CS Press, 1991.
[5] "A Framework for Software Product Line Practice, Version 4.1," Software Eng. Inst., Carnegie Mellon Univ., 2004, http://www.sei.cmu.edu/plp/frame_report/funding.htm.
[6] "Software Technology for Adaptable, Reliable Systems, Air Force/STARS Demonstration Project Experience Report," Version 3.1, vol. I, USAF Material Command, Electronics Systems Center, Hanscom AFB, Apr. 1996.
[7] J.S. Poulin, *Measuring Software Reuse.* Addison-Wesley, 1997.
[8] B.H. Barnes and T.B. Bollinger, "Making Reuse Cost-Effective," *IEEE Software,* vol. 8, no. 1, pp. 13-24, Jan./Feb. 1991.
[9] R. Malan and K. Wentzel, "Economics of Software Reuse Revisited," Technical Report HPL-93-31, Hewlett-Packard, 1993.
[10] E. Wiles, "Economic Models of Software Reuse: A Survey, Comparison and Partial Validation," Technical Report UWA-DCS-99-032, Dept. of Computer Science, Univ. of Wales, Aberystwyth, U.K., Apr. 1999.
[11] W. Lim, "Reuse Economics: A Comparison of Seventeen Models and Directions for Future Research," *Proc. Fourth Int'l Conf. Software Reuse,* pp. 41-51, Apr. 1996.
[12] S.R. Schach and A. Tomer, "Development/Maintenance/Reuse: Software Evolution in Product Lines," *Proc. First Software Product Line Conf.,* pp. 437-450, Aug. 2000.
[13] A. Tomer and S.R. Schach, "A Three-Dimensional Model for System Design Evolution," *Systems Eng.,* vol. 5, no. 4, pp. 264-273, 2002.

Amir Tomer received the BSc and MSc degrees in computer science from the Technion, Israel, and the PhD degree in computing from Imperial College, London, United Kingdom. He is the director of Systems and Software Engineering Processes at RAFAEL Ltd., Israel, where he has been since 1982, holding a variety of systems and software engineering positions, both technical and managerial. He also teaches software engineering at the Technion and other colleges. He is a member of the IEEE.

Leah Goldin received the PhD degree in computer science from the Technion, Israel, where her research focused on requirements engineering. As the CEO of Golden Solutions, she is an independent consultant specializing in software engineering, process, and quality. She has accumulated more than 20 years of experience developing embedded systems. During that period, she has fulfilled various management and technical roles, including software development, SQA, and process improvement, at Rafael, IAI, Comverse, and Nice. She currently divides her time between consulting to high-tech companies and teaching in academia; she was the head of the Software Engineering Department at the Jerusalem College of Engineering. She is a senior member of the IEEE and currently serves as the chair of the Israeli Chapter of the IEEE Computer Society.

Tsvi Kuflik received the BSc and MSc degrees in computer science and the PhD degree in industrial engineering (information systems) from Ben-Gurion University, Israel. He currently works in the Department of Management Information Systems at the University of Haifa, Israel. He has 20 years of practical experience in software and systems engineering and development, as well as practical experience in the design and development of personalized, adaptable information systems. His research interests include software engineering, artificial intelligence, and information retrieval. He is a member of the IEEE and the IEEE Computer Society.

Esther Kimchi received the BSc and MSc degrees in mathematics from the Technion in Haifa, Israel, and the PhD degree in mathematics from Tel Aviv University, Israel. She has 10 years of experience in mathematical research at Tel Aviv University, Israel, and Columbia University, New York, where she took complementary studies in computer science. She has more than 21 years of practical experience in systems and software engineering and development, project management, and company methodology definition at Tadiran Electronic Systems in Israel. She has actively participated in various conferences on software engineering and testing. She now works as an independent consultant.

Stephen R. Schach received the PhD degree from the University of Cape Town. He is an associate professor in the Department of Electrical Engineering and Computer Science at Vanderbilt University, Nashville, Tennessee. He is the author of more than 115 refereed research papers. He has written 10 software engineering textbooks, including *Object-Oriented and Classical Software Engineering*, sixth edition (McGraw-Hill, 2005). He consults internationally on software engineering topics. His research interests are in software maintenance and open-source software engineering. He is a member of the IEEE Computer Society.