Success and Failure Factors in Software Reuse

Maurizio Morisio, *Member*, *IEEE Computer Society*, Michel Ezran, and Colin Tully, *Member*, *IEEE*

Abstract—This paper aims at identifying some of the key factors in adopting or running a company-wide software reuse program. Key factors are derived from empirical evidence of reuse practices, as emerged from a survey of projects for the introduction of reuse in European companies: 24 such projects performed from 1994 to 1997 were analyzed using structured interviews. The projects were undertaken in both large and small companies, working in a variety of business domains, and using both object-oriented and procedural development approaches. Most of them produce software with high commonality between applications, and have at least reasonably mature processes. Despite that apparent potential for success, around one-third of the projects failed. Three main causes of failure were not introducing reuse-specific processes, not modifying nonreuse processes, and not considering human factors. The root cause was a lack of commitment by top management, or nonawareness of the importance of those factors, often coupled with the belief that using the object-oriented approach or setting up a repository seamlessly is all that is necessary to achieve success in reuse. Conversely, successes were achieved when, given a potential for reuse because of commonality among applications, management committed to introducing reuse processes, modifying nonreuse processes, and addressing human factors. While addressing those three issues turned out to be essential, the lower-level details of *how* to address them varied greatly: for instance, companies produced large-grained or small-grained reusable assets, did or did not perform domain analysis, did or did not use dedicated reuse groups, used specific tools for the repository or no tools. As far as these choices are concerned, the key point seems to be the sustainability of the approach and its suitability to the context of the company.

Index Terms—Survey, software reuse, empirical study.

1 Introduction

Systematic reuse is generally recognized as a key technology for improving software productivity and quality [24], possibly with a higher payoff than process improvement or process automation [3]. In many cases, object-oriented technology (OOT) is seen as an essential enabler for reuse [19] while others argue that OOT alone does not guarantee successful reuse [15].

Software reuse can take many different forms, from ad hoc to systematic; it can be based on composition or generation of code and it can involve only code or all artifacts ([25], [22]).

The reuse community initially concentrated its research on technical issues, such as repositories, tools for the search and retrieval of reusable artifacts, and programming language support. As more experience became available from industrial studies, nontechnical factors, such as organization, processes, business drivers, and human involvement, appeared to be at least as important.

Factual evidence to support the impact of these factors is still scarce or contradictory [12]. The purpose of the work reported in this paper (carried out as part of ESPRIT/ESSI project 23960 Surprise) was to survey industrial projects for the introduction of reuse and to analyze, compare, and aggregate the survey data in order to derive empirical evidence of key factors for success or failure. As far as we know, this is the only survey on reuse based on structured interviews with industrial projects.

The paper is organized as follows: Section 2 describes the methodological approach used for the study. Section 3 presents the questionnaire used in the structured interviews, while Section 4 presents the coded results. Section 5 analyzes the data set and Section 6 suggests a reuse introduction path. Section 7 summarizes related work and contrasts our results with it. Finally, in Sections 8 and 9 the validity of the study is discussed and conclusions are drawn. An appendix presents a formal analysis of the data set.

2 Research Approach

The main data source for this survey is a set of interviews with industrial projects involved in the introduction of reuse. In detail the steps followed are the following:

1. Identification and selection of projects. The European Commission has funded 288 Process Improvement Experiments (PIEs). Each PIE is a technology transfer project in which a specific software technology is applied in a company. From an empirical research point of view each project can be considered as a case study. For each PIE, a summary, a list of keywords, and a report (15 to 20 pages) are publicly available [5], [6]. Out of those 288 PIEs, 62 were short-listed on the basis of keywords and summary. That subset was analyzed in more detail, resulting in a final set of 32 PIEs that were judged to be really dealing with reuse. Most of the PIEs discarded were only introducing object-oriented techniques and claiming therefore to be doing reuse.

• M. Morisio is with the Politecnico di Torino, Dip. Automatica e Informatica, Corso Duca degli Abruzzi 24, 10129 Torino, Italy. E-mail: morisio@polito.it.
• M. Ezran is with Valtech S.A., Immeuble Lavoisier, 4 Place des Vosges, 92400 Courbevoie, France. E-mail: me@valtech.fr.
• C. Tully is with the School of Computing Science, Middlesex University, The Burroughs, London NW4 4BT, UK. E-mail: colin-tully@fsmail.net.

Manuscript received 1 Mar. 2000; revised 23 Feb. 2001; accepted 24 Apr. 2001.
Recommended for acceptance by S. Pfleeger.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111612.

2. Development of the questionnaire. The questionnaire, described in more detail in a later section, was developed through several iterations. The later versions of the questionnaire were tested by means of dry runs and modified according to the feedback received.
3. Reading of reports and interview. The 32 selected PIEs were contacted to schedule an interview at their premises. Before the interview, the interviewer(s) thoroughly read the project report and any other information available. The interview was usually with the PIE project manager, was performed by one or two interviewers, and lasted between two and three hours. The interviewee was given a copy of the questionnaire, the interviewer read questions, and took notes of answers. The interviewee was guaranteed anonymity of the data provided. It should be noted that PIEs have a contractual obligation to collaborate with other European projects, such as the Surprise project. Nevertheless, the collaboration with the interviewers was always warm and rarely seemed to be felt as a burden. In total, 19 interviews were carried out, covering 19 companies and 24 projects. The missing interviews were not performed for a number of reasons: the PIEs had only just started and had no useful results, or it was not possible to contact them, or it was impossible to arrange the interview in a convenient time frame.
4. Validation of the interview report. The interviewer produced an interview report and sent it to the interviewee for approval.
5. Coding and consistency check. The interview report was coded on the basis of a number of variables. Some of the variables had predetermined codes, so that coding was in fact performed directly during the interviews. In other cases, the variables and codes were reorganized and changed after the interviews (postformed codes). Codes were assigned on the basis of discussion and agreement between members of the Surprise team. The resulting data set is shown in Tables 1, 2, and 3.
6. Data analysis. The data set was analyzed to extract findings and trends.

3 The Questionnaire

Here, we briefly describe the questionnaire used to guide structured interviews and the reading of documents. The questionnaire was influenced by [21], [16], [30]. The main way in which the questionnaire departs from the viewpoint of those sources is that it does not endorse a specific reuse model (for instance, generative reuse instead of compositional), but tries to accept and characterize a wide variety of approaches. After the interviews, this turned out to be a good decision, as organizations used hybrid and original approaches. The definition of reuse underlying the questionnaire is:

> Software reuse is the systematic practice of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance.

Building block is purposely a loose term, to encompass a variety of software artifacts, such as code, design, requirements, test cases, code generators, etc. However, it excludes the reuse of experience or know-how. We believe this broader approach to reuse requires such a different level of infrastructure that it would require a different analysis.

Systematic practice implies a commitment and will on the part of the organization to develop the building blocks and reuse them across multiple applications. In particular, it excludes internal reuse (artifacts reused only within a project), a practice that we consider good design but not systematic reuse. It also excludes ad hoc reuse, practiced sporadically across applications, but due solely to the good practice of individual programmers.

This definition was especially used in deciding whether to include or exclude PIEs from further analysis in phase 1 of the research. Most of the PIEs excluded, while using "reuse" as a positive buzzword, did not offer any evidence of a systematic approach, and limited reuse to mean just the internal reuse of classes and objects within a single application.

The questionnaire used closed questions, except for a few open-ended ones added to collect important information which might otherwise have been overlooked. We designed it to be completed in less than two hours, to limit as much as possible the time required from busy project leaders. Actually we gained good collaboration, and most interviews lasted longer.

The questions were organized into the following sections:

- General information about the company: location, business domain, number of employees, general organization.
- Nonreuse information about the organizational unit and the baseline project. In this context, a baseline project, according to ESSI terminology, is a typical project run by the company in which reuse is applied. The baseline project is performed by an organizational unit (specific team, group, or division). Questions in this section are aimed at characterizing the software process employed by the unit (roles, documents, phases, result of assessments, certifications, etc.) and at characterizing the project (effort, duration, staff size, staff experience, size and type of software developed, languages and operating system used, analysis and design techniques used, etc).
- Reuse information. This part addresses reuse issues at company level (reuse introduction) and at organizational unit level (application of the reuse program on a specific project). Issues are grouped under the following five subsections:

  1. Organization. Motivation for reuse at the company level. Rationales, expectations, and goals of the reuse program. Management vision and commitment as regards reuse. Human factors and reuse introduction process. Training, awareness, and motivation actions performed. Barriers to change encountered.
  2. Processes. Reuse roles and processes added. Nonreuse roles and processes modified.

TABLE 1
The Data Set, State Variables

Project id	Software staff	Overall staff	Type of software production	Software and product	SP maturity	Application domain	Type of software	Size of baseline	Development approach	Staff experience
A	L	L	product-family	product	high	TLC	Technical	L	OO	high
B	L	L	product-family	product	high	TLC	Technical	M	OO	high
D	L	L	isolated	alone	middle	SE-Tools	Technical	M	OO	middle
E	L	L	isolated	alone	middle	TLC	Technical	M	OO	middle
F	L	L	isolated	alone	middle	TLC	Technical	M	OO	middle
G	L	X	product-family	process	low	Bank	Business	L	OO	middle
H	M	M	product-family	product	high	Engine Controller	Embedded-RT	L	OO	middle
I	M	X	product-family	product	middle	FMS	Technical	M	OO	high
J	M	X	product-family	product	middle	FMS	Technical	M	OO	middle
K	M	X	product-family	product	middle	ATC	Non-Embedded-RT	L	proc	high
L	M	X	product-family	product	high	TS	Technical	M	proc	high
M	M	X	product-family	product	high	SE-Tools	Technical	L	proc	middle
N	M	X	product-family	product	middle	TTC	Non-Embedded-RT	M	proc	middle
O	S	L	product-family	product	middle	Space	Embedded-RT	M	OO	middle
P	S	M	product-family	product	middle	Manufacturing	Embedded-RT	M	proc	middle
Q	S	M	product-family	product	middle	Manufacturing	Technical	M	proc	middle
R	S	M	product-family	product	middle	TLC	Embedded-RT	L	proc	high
S	S	S	product-family	product	low	Measurement	Technical	M	OO	middle
T	S	X	product-family	product	middle	FMS	Embedded-RT	M	OO	high
U	S	X	product-family	process	low	Finance	Business	L	OO	low
V	S	X	product-family	product	low	TLC	Technical	S	OO	middle
W	M	L	product-family	alone	middle	Manufacturing	Business	L	OO	middle
X	S	S	product-family	NA	low	Book-keeping	Business	S	proc	middle
Y	M	M	isolated	product	high	FMS	Embedded-RT	not available	not available	not available

3. Assets. Reuse approach (compositional versus generative approach, white box versus black box, ...). Type of assets reused, size, functions.
4. Repository. Presence and type of repository. Supporting tools and procedures. Number of assets contained.

TABLE 2
The Data Set, High-Level Control Variables

Project id	Top management commitment	Key reuse roles introduced	Reuse processes introduced	Non-reuse processes modified	Repository	Human factors
A	yes	yes	yes	yes	yes	yes
B	yes	yes	yes	yes	yes	yes
D	yes	yes	no	no	yes	no
E	yes	yes	no	no	yes	no
F	yes	yes	no	no	yes	no
G	yes	yes	yes	yes	yes	yes
H	yes	yes	yes	yes	yes	yes
I	no	no	no	yes	yes	no
J	no	no	no	yes	yes	no
K	yes	yes	yes	yes	yes	yes
L	yes	yes	yes	yes	yes	yes
M	yes	yes	yes	yes	yes	yes
N	yes	yes	yes	yes	yes	yes
O	yes	no	no	no	yes	no
P	yes	yes	yes	yes	yes	yes
Q	yes	yes	yes	yes	yes	yes
R	yes	yes	yes	yes	yes	yes
S	yes	yes	yes	yes	yes	yes
T	no	yes	yes	no	yes	no
U	yes	yes	no	yes	yes	yes
V	yes	yes	yes	yes	yes	yes
W	yes	no	yes	no	yes	yes
X	yes	NA	NA	NA	NA	no
Y	no	yes	no	no	yes	yes

5. Metrics. Metrics and cost models used.

4 The Data Set

After the interviews were performed, the Surprise team members coded the interview results to form a data set (phase 5 of the research) presented in Tables 1, 2, and 3.

In this section, we define the meaning of variables and comment on their values.

Each data point (a row in the table) corresponds to one project. When a PIE was performed in a company, reuse might have been experimentally adopted on one or more projects. Therefore, in the table there are 24 projects, or data points, for 19 PIEs in 19 different companies. In each table, shaded rows depict projects that failed, while unshaded rows correspond to successful projects. The definition of failure and success will be presented later.

Each data point has several attributes, or variables. They are further divided between state variables[1] (attributes over which a company has no control, such as size and application domain) (Table 1) and control variables (attributes a company can control, such as commitment of management, modifications to the process, reuse approach, etc.) (Tables 2 and 3).

The meanings of the state variables (Table 1) are given below.

Project id: identifier of the project.

Software staff: S (small)—from 1 to 50 people; M (medium)—from 51 to 200; L (large) more than 201. Note that only staff involved in software development are considered.

Overall staff: S (small)—from 1 to 50 people; M (medium)—from 51 to 200; L (large) from 201 to 500; X (extra large) more than 501. Here, all staff are considered.

Software production: isolated—the company develops projects that have little or nothing in common; product family—the company develops a software product that evolves over time, and/or is more or less adapted for each customer.

Software and product: product—the software is embedded in a product; alone—the software constitutes a standalone product; process—the software is embedded in a process.

SP (software process) maturity: high—CMM level 3 or higher (actual or estimated); middle—ISO 9001 certification or

1. One can argue that, for some state variables such as staff size and staff experience, a company actually has control and can change them. In these cases we use time as a discriminant: if a variable requires more than a few days or weeks to be changed it is considered as a state variable.

TABLE 3
The Data Set, Low-Level Control Variables

Project id	Reuse approach	Work products	Domain analysis	Origin	Independent team	When assets developed	Qualification	Configuration management	Rewards policy	# assets
A	tight	D+C	yes	ex-novo	yes	before	yes	yes	no	51-100
B	tight	D+C	yes	ex-novo	yes	before	yes	yes	no	51-100
D	loose	C	no	as-is	no	before	no	no	yes	21-50
E	loose	C	no	as-is	no	before	no	no	yes	21-50
F	loose	C	no	as-is	no	before	no	no	yes	21-50
G	loose	C	no	reeng	no	justintime	yes	yes	no	51-100
H	tight	R+D+C	no	reeng	no	justintime	no	yes	no	51-100
I	loose	D+C	no	reeng	no	justintime	no	no	no	51-100
J	loose	D+C	no	reeng	no	justintime	no	no	no	51-100
K	tight	R+D+C	yes	reeng	no	justintime	yes	yes	no	100+
L	tight	R+D+C	yes	reeng	no	justintime	yes	yes	no	51-100
M	tight	R+D+C	yes	reeng	no	justintime	yes	yes	no	100+
N	tight	R+D+C	yes	reeng	no	justintime	yes	yes	no	51-100
O	loose	R+D+C	no	ex-novo	no	before	yes	yes	no	1-20
P	loose	R+D+C	yes	reeng	no	justintime	yes	yes	no	100+
Q	loose	R+D+C	yes	reeng	no	justintime	yes	yes	no	100+
R	loose	C	no	reeng	no	justintime	yes	yes	no	1-20
S	tight	C	no	ex-novo	no	justintime	yes	yes	no	100+
T	loose	C	no	reeng	no	justintime	yes	yes	no	1-20
U	tight	R+D+C	no	reeng	no	justintime	no	yes	no	100+
V	tight	C	no	reeng	no	justintime	yes	yes	no	1-20
W	tight	C	yes	reeng	no	justintime	no	no	no	1-20
X	NA	NA	NA	NA	NA	NA	NA	NA	no	NA
Y	loose	C	no	as-is	no	before	no	no	no	100+

CMM level 2 (actual or estimated); low—no ISO9001 certification or CMM level 1 (actual or estimated).

Application domain: TLC—telecommunications; FMS—flight management systems; ATC—air traffic control; TS—train simulation; TTC—train traffic control; Bank; Book-keeping; Measurement—management and control of measurement environment; Space—aerospace applications; Manufacturing; SE-Tools—software tools.

Type of software: Embedded-RT—embedded, real-time; Non-Embedded-RT—nonembedded, real-time; Technical—nonembedded, nonreal-time, small DBMS, important control part; Business—nonembedded, nonreal-time, important DBMS, limited control part.

Size of baseline (size of the software project on which reuse was applied): S (small)—<10KLOC and 10 person-months effort; M (medium)—10-100 KLOC and 10-100 person-months; L (large)—100-500 KLOC, more than 100 person months.

Development approach (analysis and design approach used in the project): OO—object oriented; proc—procedural.

Staff experience: high—on average > five years; middle—two to four years; low—one year or less.

Note: "NA" (project X) stands for "not applicable." The same abbreviation is also used in Tables 2 and 3.

The projects in the data set represent a large variety of situations.

Software staff ranges from small to large. *Overall staff* ranges from small to very large. Very small, small, and medium companies are fairly represented, the smallest being a 10-person software house. As far as we know, small companies have never reported their reuse experience in the literature. It is reasonable to suppose that this was due not to their absence from the arena, but to not having the resources to publish.

If we consider the variable *software production*, "product family" outnumbers "isolated" by 20 to 4. "Product family" means that the company produces applications that resemble one to another, either a product that evolves over time, or an application customized for different customers, or both. "Product family" does not mean that the company is using a domain engineering or product line approach. It is apparent that the majority of companies recognize the commonality in their products and explore reuse as a consequence.

In *software and product*, only four cases out of 24 deal with software not embedded in products or processes. This could be interpreted as consistent with the growing diffusion of embedded software in products and processes. Or it could depend on the fact that, compared to standalone software, software embedded in a range of products or processes tends to require an appropriate balance between fundamental commonality and detailed variability, thus creating favorable conditions for reuse.

Considering *software production* and *software and product* together, 20 out of the 24 cases are product family, and 17 are product-embedded. The straightforward conclusion is that organizations tend to identify product families where they produce product-embedded software and, that, the combination of these two related characteristics seems to offer natural conditions for reuse.

As far as *software process maturity* is concerned, only five projects out of 24 were developed in organizational units with a low maturity.

Application domain covers a wide variety of domains, with a prevalence of telecommunications (six cases).

Type of software focuses on the characteristics of the software developed and ranges from embedded real time to database intensive. Here, the prevalence (half the cases) is in "technical" applications, i.e., nonembedded, nonreal-time, with limited or no database and an important algorithmic or control part. Overall, 20 of the 24 projects are what could be described as of a software engineering nature, with only four of an information systems nature. The cause of this bias could be intrinsic to the nature of the software in the two domains, or intrinsic to the differing software development cultures, or a combination of the two.

Size of baseline, i.e., the size of the project where assets are reused contains a large majority of "medium" and "large" projects, where medium is defined as more than 10KLOC size and/or more than 10 person-months effort. No "very large" (more than 500KLOC) projects appear. Companies have judiciously applied reuse initially on real-life projects, excluding both toy projects and very large ones.

Development approach. The object-oriented approach covers the majority of cases, but the procedural approach is well represented (eight out of 24).

Staff experience. Only one project was performed by a group of beginners (average experience below one year). Companies generally assigned projects involving reuse to their more experienced staff.

In summary, the data set contains large and small companies, working in a variety of domains. Most of them produce software applications embedded in products or processes, with high commonality between applications. Most of them have reasonably mature processes and assign experienced staff to reuse projects. Both procedural and object-oriented development approaches are well represented.

The state variables of this data set satisfy, in theory, many of the prerequisites for successful reuse. In practice, however, nearly 40 percent of the projects (the shaded rows in Table 1) were judged to have resulted in failure.

To understand why this happened we analyzed the control variables—those characteristics that are under the influence of a company's own decision processes.

We divide control variables into two categories: high-level and low-level. High-level control variables correspond to decisions that (whether taken explicitly or by default) subsequently influence low-level control variables. They are shown below in Table 2 and subsequently defined. As before, shaded rows represent projects that are judged to have failed.

Project Id: identifier of the project.

Top management commitment: yes—top management of the company had a clear commitment to introducing and sustaining reuse; no—top management did not show that commitment, so that reuse was initiated bottom-up from middle managers or technical staff.

Key reuse roles introduced: yes—at least one reuse role (such as reuse program manager, asset owner, library manager, asset producer) was introduced; no—no reuse roles were introduced.

Reuse processes introduced: yes—at least one reuse-specific process (such as domain analysis, qualification, classification) was introduced; no—no reuse process was introduced.

Nonreuse processes modified: yes—at least one nonreuse-specific process (such as requirements analysis, design, testing.) was modified; no—no nonreuse-specific processes were modified.

Repository: yes—assets were stored in an asset repository, supported by a tool (not necessarily a dedicated reuse repository tool); no—no asset repository was established.

Human factors: yes—human factors were considered and dealt with via (for instance) awareness, training, and motivation actions; no—human factors were not considered.

Values of the high-level control variables represent the combination of key high-level management decisions about a reuse program. For instance, in case A it was a top management decision to embark on reuse; a reuse group was set up, both to produce reusable assets and to support reusers; the software development process was modified appropriately; a repository for assets was set up; and training and awareness actions were started to support the transition.

In case T, in contrast, the reuse program started as an initiative by middle-level managers. A repository was set up, a library manager was appointed and legacy work products were placed in the repository. However, no modifications were made to preexisting processes and no actions were taken to advertize the reuse program.

We emphasize that "Top management commitment = no" means that the change initiative came from some combination of individuals at middle management and practitioner levels. As will be shown in more detail later, in these cases middle management and practitioners on their own are not able to support and enforce all the changes necessary for success.

While a "no" on a high-level control variable means no action (whether by explicit decision or by default), a "yes" may represent a whole range of possible actions or choices. Low-level control variables represent some of them, or decisions at a lower level of detail on how to implement a reuse initiative. The low-level variables defined here do not cover all possible approaches, but cover the cases we analyzed.

These variables are shown in Table 3 and are defined subsequently.

Project Id: identifier of the project.

Reuse approach: loose—reusable work products are loosely coupled and can be reused in isolation; tight—reusable work products are tightly coupled, a group of them is reused at each time.

Work-products (type of reused assets): C—code, D—design, R—requirements, C+D code and design, etc.

Domain analysis: yes—domain analysis was performed; no—no domain analysis was performed.

Origin: ex-novo—assets are developed from scratch; reeng—assets are developed by reengineering existing work-products; as-is—assets are existing work-products, reused without modification.

Independent team: yes—a team, independent from development projects, develops reusable assets; no—development projects both develop and reuse assets.

When assets developed: before—assets are developed before a reusing project needs them; just-in-time—assets are developed just before a reusing project needs them.

Qualification: yes—assets undergo a defined qualification process to be declared reusable; no—no defined qualification process.

Configuration management: yes—reusable assets are under configuration management and change control; no—no configuration management and change control for assets.

Rewards policy: yes—a rewards policy to promote reuse is in place; no—rewards policy not in place.

# assets: number of reusable assets in the repository.

The combination of values for these low-level control variables represents a certain approach to the implementation of reuse. For instance, in case A a dedicated group produces object-oriented frameworks with their documented designs. These assets are produced from scratch, some time before projects need them. Work-products are subjected to a qualification process and are under configuration control (configuration management = "yes").

Case R works quite differently. When a project developing an application identifies a legacy subprogram that could be reused by the same project and by others, it reengineers it for reusability and quality.

The *reuse approach* variable requires further elaboration.

All projects have used a compositional approach, with a variety of choices of which work products should be reusable and what mechanisms to use in assembling them. However, we can make a distinction between tight and loose approaches as far as the relationship among reused work products is concerned.

- Loose. Reusable work products are independent and can be reused in isolation. No common architecture is defined.
- Tight. Reusable work products are designed to be closely related; reuse of a single work product involves reusing a wider set of work products. This approach is normally associated with object-oriented and framework technology, but some companies used it with traditional procedural technology also.

The tight approach consists in engineering a generic product that mirrors the specific business of the company. The generic product is then instantiated and customized for each different customer or application. Reusing a generic product means that a standard architecture is defined and its use enforced. The tight approach does not exclude the loose approach, which can also be used when suitable. We found several different forms of tight approach.

- Domain analysis, frameworks (projects A, B, W). After a domain analysis effort, one or more frameworks are built (A, B) or reengineered from legacy (W) and maintained. The reusable asset unit is the framework, with the related documentation. Object-oriented technology is used extensively.
- Product baseline (projects H, K, L, M, N, U). The generic product is built with procedural technology, and maintained over time as a sequence of versions in a configuration management system. Specific instances for customers are derived from the product baseline, by additions, deletions, and modifications. The product baseline is built using domain analysis (K, L, M, N) or less formally using the know-how of senior designers (H, U). The reusable asset is the product baseline, and the related documentation.
- Composition language (project S). One company has defined source-code work products and a language for assembling them, which could be called a hybrid generative and compositional approach. The reusable asset is the subset of work products selected by the directives of the composition language.

The dependent variable for this study is the success of the PIE. The dependent variable does not appear as a column in the tables. Instead, rows with failed projects are shaded and rows with successful projects are not shaded.

Success is judged according to three criteria: the continuation of the reuse program after the PIE is over, the soundness of the approach, and the fact that assets were actually reused. Initially, we tried to use a more quantitative measure of success or failure. However, it turned out that only one company was collecting such measures, such as the return on investment for the reuse program. On the other hand, in most projects listed as failures, the failure itself was evident to the interviewees and openly admitted and discussed. It should be noted that, for the ESSI initiative, a successful project is a project that reports clearly on its results; these are equally well received, irrespective of whether they are positive or negative.

5 Analysis of the Data Set

The analysis of the data set built up from the questionnaires must take into account two important constraints: the number of data points is very limited for significant statistical results and the attributes collected all have categorical scales (nominal or ordinal).

On the other hand, this study is a mixed qualitative and quantitative study. According to Seaman [33], qualitative information is expressed using pictures and words, quantitative information with numbers and symbols. Although the misconception *qualitative is subjective, quantitative is objective* is common, the subjectivity or objectivity of information is orthogonal to its qualitative or quantitative status. Qualitative information is richer, but requires suitable techniques for its analysis.

The interviews yielded deep knowledge on the reuse initiatives and provided interview notes and answers to the questionnaire (mostly qualitative information). The coding process produced the data set as already shown in Tables 1, 2, and 3 (quantitative information). We will use both types of information to perform an exploratory analysis and to find relationships in the data set that can be used as research hypothesis in further studies.

We organize our analysis in three steps.

1. Successes and Failures. We looked for a correlation between the independent variables and the dependent variable (success/failure).

   Given the limited number of cases, it is not possible to consider all independent variables in the analysis. We focused our attention on control variables. While all of them represent a decision, some of those decisions temporally and/or logically precede others. For instance, a decision to introduce reuse processes both logically and temporally precedes a decision about whether to perform domain analysis and about when to develop assets. Accordingly, we partition control variables between high-level and low-level, according to the temporal/logical order of the corresponding decisions.

   The analysis here is limited to state and high-level control variables. The analysis shows that *not* addressing two or more high-level control variables led to failure. Of the state variables, only *Type of software production* has an impact.
2. Failures. In most cases, failure in the reuse program was clear to the interviewees, who openly discussed the causes. These discussions went way beyond the points in the questionnaire and could, in most cases, identify the root causes of failure. We present a root cause analysis for each failure and derive two common failure scenarios.
3. Successes, State, and Low-Level Control Variables. The analysis at point 1 above shows that all successes address similarly a number of common issues (high-level control variables), while all state variables except one have no influence. However, as we know from qualitative data from interviews, successful projects solved the reuse equation in a variety of ways. This is captured by the variety of values taken by the low-level control variables. Is there a regularity in the diversity of approaches to reuse? Do state variables influence the approach to reuse? Here, we restrict the analysis to successful cases, state variables, and low-level control variables.

5.1 Successes and Failures

We look for a correlation between independent variables (restricted to state variables and high-level control variables, see Tables 1 and 2) and the dependent variable (success or failure). We recall that failures are highlighted, in Tables 1, 2, and 3, with shaded rows.

An examination of Table 2 shows that successful projects have a "yes" value for all high-level control variables, the only exceptions being projects U and W. Projects that failed have three or more "no" values.

In the appendix, a formal tool, correlation tree analysis, is used to analyze the data. The result is similar: high-level control variables appear to have more predictive importance than the rest. Classification trees also point out that one state variable, *Type of software production*, has predictive importance.

By combining the results from classification tree analysis, the analysis of failures, and further insight gained from the interviews, we argue that all high-level control variables are important for a successful reuse program. *Introducing reuse processes* and *Modifying nonreuse processes* are the key points for successfully producing and consuming assets. *Top management commitment* is the prerequisite for successfully designing and enacting process change. At the other end of the spectrum, *Human factors* must be addressed to sustain process change from the bottom up.

Introducing key reuse roles and setting up a *repository* are not sufficient for successful reuse. This is not to say that they are unimportant. The key point is that both represent relatively minor changes in a company, which can be accomplished with partial management support. As a result, they are done in most projects, but cannot, by themselves alone, induce the major changes needed.

Of the state variables, only *Type of software production* has an effect. Most of the cases with value "product family," in contrast with none of the cases with value "isolated," were successful. We do not think this is enough to state that "isolated" cases are not suitable for reuse, however, since there are only four "isolated" cases in the sample and none of them took enough account of the issues addressed by the high-level control variables. In fact, "isolated" cases could probably achieve success; but, because they have a lower reuse potential, they should take even more careful account of the high-level control variables.

Size of the company, as measured by both *Software staff* and *Overall staff*, does not appear to be a conditioning factor. However, size impacts indirectly on two things.

- The ease or difficulty for achieving commitment of top management, and its propagation to lower hierarchical levels. Successful smaller companies (projects P, Q, R, S) have the advantage of easier communication of information (for instance, information about reusable assets, domains, and projects is more easily shared among the staff) and easier building of consensus for the reuse program (the program is initiated when the occupier of a prominent role in the company—owner, director, technical lead—decides accordingly). Failure in two projects (O, T) happened in two small software organizations belonging to large nonsoftware companies. In those

Root causes of failure	Misconceptions (reuse = repository, reuse = OO)
Secondary causes of failure	No non-reuse-specific processes modified (except rewards) No reuse-specific processes installed No training/awareness actions

Fig. 1. Failure scenarios for projects D, E, and F.

cases, commitment from management of the small software organization was not sufficient to remove obstacles at the upper level.

- The reuse organization. Successful smaller companies (projects P, Q, R, S) find leaner organizations are adequate to support reuse processes (generation of assets, qualification, maintenance, domain analysis). Roles dedicated to reuse have necessarily to be part-time. The production of assets is made on demand. Failure in a small company (project X) was due to defining a reuse infrastructure that was too complex, with complex procedures and full-time roles.

Other state variables that do not appear to be predicting factors nevertheless merit some discussion.

Software process maturity: As already discussed, the companies considered have reasonably mature processes. We may assume that process maturity is a useful but not sufficient factor in achieving success.

Type of software: In two cases (O, T), embedded-RT software (and, specifically, sudden changes to hardware and performance constraints) was related to failure. Conversely, there are three successful cases (H, P, R) with embedded-RT software. We tend to believe that the distinguishing factors here are hardware changes and performance constraints, which, in many cases, are associated with embedded-RT software.

Development approach: The misconception that object-orientation on its own is enough to guarantee successful reuse appears in some of the failure scenarios. Further, we recall that many other projects (30 out of 62) shared this misconception and were therefore not included in the analysis.

5.2 Failures

Now, we analyze the projects that failed. We derive two failure scenarios (Figs. 4 and 5) and we present the qualitative evidence supporting them (Figs. 1, 2, and 3).

5.2.1 Projects D, E, F

These projects introduced reuse with two changes to the current process and organization.

First, they introduced a company-wide, intranet-based repository, along with a role to manage the repository responsible for documentation and version control of assets.

Root causes of failure	No deep management commitment Misconceptions (reuse = repository)
Secondary causes of failure	No non-reuse-specific processes modified No production of assets No awareness actions

Fig. 2. Failure scenarios for projects I, J, and Y.

Root causes of failure	Embedded RT system context (memory and speed constraints) Multi-contractor / multi-company project (no ownership of choices in hardware and requirements)
Secondary cause of failure	Reusable assets produced but then not used

Fig. 3. Failure scenarios for projects O and T.

Second, they introduced a complex policy to reward both producers and consumers of assets. An asset is placed in the repository without quality checks and the producer is rewarded for each actual reuse. The rationale behind this approach is to disturb as little as possible the ongoing business, while trying to initiate an internal market of assets. No mandatory process changes are made, either for the consumers or the producers of assets. Also, this approach assumes that object-orientation in itself leads to effective reuse. In fact, however, few assets are produced and few, if any, are reused. Most of them are for small functions. The lesson from these projects is that installing a repository and a rewards policy alone are not enough to achieve systematic reuse.

5.2.2 Projects I, J, Y

In case Y, the company set up an automated repository and filled it with legacy assets (mostly small-grained functions not specific to a domain) from past projects. In cases I and J, a loosely formalized repository was set up to contain a limited number of purpose-built assets. No modification of nonreuse processes took place and no systematic production of assets was initiated. This was perhaps a partial improvement since the repository in case Y was paper-based. However, the net result was that no assets were ever reused.

Not modifying nonreuse processes, and insufficiently publicizing the repository and the reuse initiative, were the immediate causes of failure.

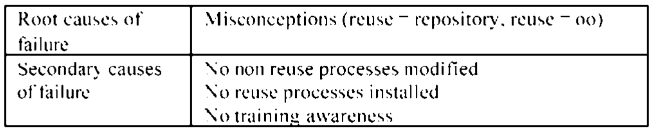
The root cause could be a weak involvement of top management in the reuse initiative. Hence, a missing will and power to change existing processes.

5.2.3 Projects O, T

These two projects, although in different companies, share several common characteristics that appear to be in both cases major causes of failure. They produce embedded real-time software and they are subcontractors in bigger projects (Fig. 3).

Both projects tried to address changes to reuse processes and the introduction of nonreuse processes. However, they were not able to achieve these changes since they were subcontractors and they did not own all the processes. Moreover, because of performance constraints imposed on the embedded real-time software, they could not use the well-known technique of decoupling software from hardware using layers. As a result, reusable assets were produced, but could never be reused because of changes to requirements, both in functionality and hardware. Finally, the reuse initiatives were abandoned. Both projects agreed reuse could have been achieved in their context, but only if started at the level of the main contractor.

The lesson here is that a subcontractor can rarely decide to go for reuse independently of the main contractor,



Root causes of failure	Misconceptions (reuse = repository, reuse = oo)
Secondary causes of failure	No non reuse processes modified No reuse processes installed No training awareness

Fig. 4. Failure scenario 1.

especially in the delicate case of embedded real-time software. Embedded real-time software is not a cause of failure per se, however, as demonstrated by other successful cases (projects H, P, and R).

5.2.4 *Project X*

The reuse initiative was interrupted before completion because the company recognized it could not achieve the objectives set. On the one hand, the objectives were too ambitious, especially for a very small company like the one where the project was performed. On the other hand, factors extraneous to the reuse initiative intervened, such as key personnel turnover and crises in project management.

Project X is listed in the data set for completeness, but it is not used in the data set analysis since several fields are incomplete due to its interruption.

Failure Scenarios. Among the failures that we have analyzed above, we can recognize two similar sets of projects. Projects D, E, F, I, J, and Y failed because of misconceptions (Fig. 4). Projects O and T failed because of the context—embedded real-time software and a multi-company/multicontractor environment (Fig. 5). We summarize those two sets by hypothesizing two failure scenarios. We do not believe that they are exhaustive: more could be proposed by observing other projects. Project X is not considered in these scenarios. Project X is an example of failure with no specific reuse-related causes.

- Failure Scenario 1 (supported by cases D, E, F, I, J, and Y).

All these failures are, in some way, related to the misconception that installing a repository is the key point in a reuse program, and/or that object-orientation automatically leads to successful reuse.

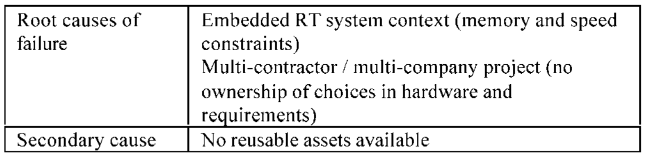
Sometimes management did not support fully the reuse initiative (I, J). These points are related, setting up a repository is a relatively easy and nonintrusive task that can be performed offline without interfering deeply with everyday processes. On the contrary, serious changes require top management commitment. However, top management commitment exists on half the failure cases and is absent in the other half. So, we can't conclude that its absence on its own inevitably leads to failure.

- Failure Scenario 2 (supported by cases O and T).

This scenario corresponds directly to the description already given for projects O and T.

5.3 Successes, State, and Low-Level Control Variables

All the successful reuse initiatives addressed a common set of issues, as captured by the high-level control variables, and that was a key element of their success. But, the precise



Root causes of failure	Embedded RT system context (memory and speed constraints) Multi-contractor / multi-company project (no ownership of choices in hardware and requirements)
Secondary cause	No reusable assets available

Fig. 5. Failure scenario 2.

way of addressing that set of issues, as captured by low-level control variables, was quite diverse.

The first question is whether some recurring pattern can be recognized among low-level control variables. If yes, is this pattern related to the state variables? For instance, do all small companies use pattern X, while large companies use pattern Y?

To explore this, we analyzed a reduced version of the data set. We limited our analysis to successful projects and tried to find relationships between state variables and low-level control variables. We exclude high-level control variables from the analysis because they were addressed substantially in the same way by all the successful projects.

Further, we exclude *Type of software production* and *Rewards policy* (all successful projects have the same value for these variables) and also exclude *Application type* and *Size of baseline.*

Table 4 shows the reduced data set. The thick vertical line divides state variables (on the left) and low-level control variables (on the right).

Visual analysis of the data set is now more complex. We will also use cluster analysis (see the appendix for details) as an analysis tool. Cluster analysis applied to low-level control variables discriminates a cluster comprising cases A and B from the data set, and another group comprising the rest. A and B are discriminated by the values for the variables *Origin*, *When asset developed*, and *Independent team*. They are the only cases featuring an independent group to develop reusable assets, starting from scratch, with a tight approach. These two cases, both from the same company, represent a sophisticated approach to reuse, requiring more reorganization and investment. We could name this approach "sophisticated."

The other cases (we could call them "pragmatic") have no independent team and develop assets just in time in most cases by reengineering.

From the data, we have it is not possible to evaluate which approach produced better results. One can argue that this may not be important since each approach was adapted to the context and produced positive results.

Digging more into the "pragmatic" cases, G and R share a loose *reuse approach,* limit themselves to code *work products* and do not use *domain analysis*. However, there is a lot of variation in values on these three variables and it is hard to identify meaningful patterns.

Overall, merging this analysis of data and insight from interviews, we argue that companies had two approaches to implementing reuse, a more sophisticated and a more pragmatic one. Inside the pragmatic approach, many choices are possible. All approaches can work, provided they are adapted to the context.

As far as state variables are concerned, it is hard to find meaningful patterns. This result is confirmed by cluster

TABLE 4
State and Low-Level Control Variables, Success Cases

Project id	Software staff	Overall staff	Software and product	Software Process maturity	Type of software	Development approach	Staff experience	Reuse approach	Work products	Domain analysis	Origin	Independent for team	When asset developed	Qualification	Configuration management	# assets
A	L	L	product	high	technical	OO	high	tight	D+C	yes	ex-novo	yes	before	yes	yes	51-100
B	L	L	product	high	technical	OO	high	tight	D+C	yes	ex-novo	yes	before	yes	yes	51-100
G	L	X	process	low	business	OO	middle	loose	C	no	reeng	no	just in time	yes	yes	51-100
H	M	M	product	high	embedded-RT	OO	middle	tight	R+D+C	no	reeng	no	just in time	no	yes	51-100
K	M	X	product	middle	non-embedded-RT	proc	high	tight	R+D+C	yes	reeng	no	just in time	yes	yes	100+
L	M	X	product	high	technical	proc	high	tight	R+D+C	yes	reeng	no	just in time	yes	yes	51-100
M	M	X	product	high	technical	proc	middle	tight	R+D+C	yes	reeng	no	just in time	yes	yes	100+
N	M	X	product	middle	non-embedded-RT	proc	middle	tight	R+D+C	yes	reeng	no	just in time	yes	yes	51-100
P	S	M	product	middle	embedded-RT	proc	middle	loose	R+D+C	yes	reeng	no	just in time	yes	yes	100+
Q	S	M	product	middle	technical	proc	middle	loose	R+D+C	yes	reeng	no	just in time	yes	yes	100+
R	S	M	product	middle	embedded-RT	proc	high	loose	C	no	reeng	no	just in time	yes	yes	1-20
S	S	S	product	low	technical	OO	middle	tight	C	no	ex-novo	no	just in time	yes	yes	100+
U	S	X	process	low	business	OO	low	tight	R+D+C	no	reeng	no	just in time	no	yes	100+
V	S	X	product	low	technical	OO	middle	tight	C	no	reeng	no	just in time	yes	yes	1-20
W	M	L	alone	middle	business	OO	middle	tight	C	yes	reeng	no	just in time	no	no	1-20

analysis (see appendix). So, we cannot find any correspondence between groups identified by state variables, and groups (sophisticated and pragmatic) identified by low-level control variables.

A likely explanation is that the data set is too limited to find any correspondence. The variables in the data set do not capture information that might be necessary to establish correspondences. Going back to direct experience from the interviews, we argue that a relationship could exist, but it is not captured by the variables in the data set. Cases A and B chose the sophisticated approach because management decided to invest deeply in reuse as one of the key technologies for the success of the company. As a result, more resources were available and the sophisticated approach became feasible. More resources could be allocated also because of the large size of the company.

As for the other cases, the pragmatic approach was chosen because of a more prudent approach to reuse. Here, size is a factor. Smaller companies face real constraints in allocating resources that force them to find leaner organizations, with part-time reuse roles, and production of assets on demand.

6 A Reuse Introduction Decision Sequence

We summarize the results of the previous analysis by presenting a decision sequence. The decision sequence merely tries to explain the cases in the data set and does not claim scientific validity as a prediction tool for new cases. However, it highlights issues that should be considered when starting a reuse program. The reader can find in [25] a more detailed analysis of four successful cases.

1. Reuse potential. Evaluate the reuse potential, which is much higher when similar software products are produced over time (*Type of software production* = product family). In practice, this is not an easy task,

as it involves identifying the functions likely to be reused and the number of times they could be reused within a given time period. Several techniques under the heading of domain analysis and product lines [35], [1] have been proposed to guide this task.

2. Reuse capability. Get commitment of top management to obtain resources and power to:

- Change nonreuse-specific processes.
- Add reuse-specific processes.
- Address human factors.
- Set up a repository.

The points above are not in a significant order; they should all be addressed. When two or more of them are not addressed, a failure is likely. Adding reuse-specific processes normally implies defining and assigning key reuse roles, so the latter is an additional implicit requirement. The factor common to all of them is *change* and, so, the prerequisite is commitment from top management. Another prerequisite is knowing what the processes are. Here, two factors are involved: size of the organizational unit and process maturity. Small size and high process maturity clearly help.

Common misconceptions (OO or setting up a repository automatically means successful reuse) lead to the likelihood of overlooking the importance of addressing all the points above.

Check ownership of processes and requirements, especially in the case of embedded real time software. Changing nonreuse-specific processes and adding reuse-specific processes will be much more difficult when ownership of those processes lies elsewhere, i.e., when subcontracting is involved.

3. Reuse implementation. Each of the points above has to be addressed through further lower-level choices.

- **Change nonreuse-specific processes**. Requirements definition and analysis, high-level design, and testing, all require specific changes to take into account the availability of assets. Project management is impacted too, as far as scheduling, costs and productivity are concerned.
- **Add reuse-specific processes**. Domain analysis might or might not be used to drive the identification of reusable assets. Assets could be smaller or larger in size, including design and requirements or not. They could be developed from scratch, or reengineered from legacy. They could be produced and maintained by a specific group, or by application projects; before projects need them, or just in time for the first use.
- **Address human factors**. One or more techniques (such as training, awareness events, discussion groups, newsletters) can be used. Reward systems alone are not sufficient.
- **Set up a repository**. A specific tool, add-ons to the configuration management system, or the plain configuration management, are all possible options.

The availability of resources in the company, usually related to its size, should be carefully considered in arriving at decisions that can be sustained. Provided the approach is sustainable, integrated, and adapted to the context, any combination of choices is acceptable.

Overall, successful cases always tried to minimize change. They retained their existing development approach and chose reuse technology to fit that approach. They often used their existing configuration management tool for the repository. The advantage here lies in introducing as few changes at a time as possible and in building on existing knowledge, skills, and tools. The central question becomes: What is worth changing and what is not? Successful cases teach us that change should focus primarily on processes and roles. Development technology and supporting tools can be changed later, if necessary.

Changes to processes and roles should be affordable. The companies' choices varied greatly, yet if we relate them to their size and available resources, a logic appears. For instance, only bigger companies can sustain a separate reuse group. The same applies to domain engineering, a process that only few can afford. Finally, assets are developed in advance by the bigger companies, while the others develop assets just in time for the first reuse.

7 Comparison with Related Work

We analyze here previous studies published in the literature about the effects of, or prerequisites for, reuse in industrial contexts. We group them into two categories: surveys of multiple companies via mailed questionnaires and experience reports from individual companies.

7.1 Mail Surveys

Rine and Sonneman [32] did a mail survey in 1995. A questionnaire was sent out and generated 109 responses, representing 99 projects in 83 organizations, most of them in the USA. No information on the size of companies or projects was reported. Software reuse capability was measured according to the percentage of components (bought or developed) reused in a project, and according to the percentage of components developed in one project and reused by other projects. The main findings of the study are that factors like product line practice, standard data formats, common software architecture, domain engineering, commitment from management, and stress on reusing high-level software artifacts versus just code, positively influence software reuse capability. In turn, software reuse capability positively influences quality and productivity.

Lee and Litecki [24] mailed, before 1993, a questionnaire on reuse to the Ada community (defined as the subscribers to three Ada journals) and obtained 75 responses. The paper does not report how many projects or companies these answers represent. The dependent variable is reuse rate (size of reused Ada code/size of all code). The paper does not clarify if this measure is computed on a sample project, or is an average across all company projects. Domain knowledge, reuse experience, maturity of OOD tools, maturity of repository, executive directors' concern, level of reuse program, number of reuse engineers, and size of repository are identified as positive influences on the dependent variable. Both Ada experience and maturity of Ada tools negatively influence the dependent variable.

Frakes and Fox [10] mailed a questionnaire in 1991-1992. They obtained 113 responses from 23 companies and six universities, representing 28 US organizations, from small to very large. The dependent variable is the average level of reuse of code in the organization. The results are that the dependent variable is not affected by programming language, CASE tools, "not invented here" syndrome (or the fact that people prefer to redo from scratch instead of reusing), experience, rewards, legal problems, repositories, organizational size, quality concerns, or reuse measurement. On the other hand, the dependent variable is affected by reuse education, type of industry (telecom, aerospace, etc.), perceived economic feasibility, and availability of high quality assets.

In the same questionnaire, Frakes and Fox [11] asked respondents to identify the most common problems encountered in reusing assets. The question assumes a component-based approach to reuse and a working reuse program. The resulting ranking is: no attempt to reuse (32 percent); components do not integrate (22 percent); component not understood (21 percent); component not valid (19 percent); component does not exist (18 percent); component not found (12 percent); component not available (7 percent).

7.1.1 Comparison

All the above studies use some variation of the "reuse level" measure in evaluating the success of a reuse program. Reuse level has the advantage of being easy to calculate or estimate, especially in the context of a mailed questionnaire. However, it can be unfair in some cases and too generous in others, depending on the reuse approach. A 70 percent reuse level could be a bad result in the case of a product line approach, while a 30 percent level could be a remarkable success for an approach based on a library of small assets. Further, reuse level migth or might not take account of internal reuse. Finally, even when formally measured, reuse level is far from being a standard measure and results in very different counting in different companies, or even different projects in the same company [30].

For those reasons, we have considered reuse level as one of the indicators of success for a reuse program, but we have combined it with others. Further, the above studies only consider reuse programs when they are up and running. In our case, we also study the previous phase, the introduction of reuse. Frakes and Fox [10] consider failures, but in the sense of inability to find a reusable component.

The other important difference between the above studies and the study we are reporting here is the research method used. Mail surveys may generate large numbers of responses, but those responses cannot be controlled and verified as is possible in structured interviews. Further, interviews make it possible to dig deep in specific areas and achieve a fuller understanding of the problem at hand.

Keeping those differences in mind, we now compare the results of the various studies.

In agreement with Rine and Sonneman [32], we find that management commitment is a significant factor. We go further and argue that management commitment is an essential prerequisite to support the range of changes needed in implementing a reuse program. They report also product line practice, common software architecture, and domain engineering as factors that increase the reuse level (or reuse capability in their terms). These factors are very similar to our *Reuse approach* = "tight" factor. Cases A, B, H, K, L, and M have a tight reuse approach and report high reuse levels (50 to 90 percent). Cases P and Q have a loose approach and report low reuse levels (around 15 percent). We argue that a high reuse level is in some sense intrinsic to this approach. An economic analysis (for instance a return on investment analysis) would be a suitable tool to evaluate which approach is more suitable in a given context.

Lee and Litecki [24] report domain knowledge and reuse experience as factors. In our case, all projects were introducing reuse for the first time, so reuse experience is presumably low by definition. We observe that *Staff experience* (i.e., overall experience excluding reuse) has a high or medium value in all cases with the exception of one low value, which suggests that it is often a significant factor. Size of repository is also a factor in their study. In all our cases, a repository existed and its size was limited (fewer than 200 assets). Two companies that failed had among the largest repositories. We conclude that having a repository is useful but not itself a guarantee of success and that large size may be inversely correlated with success.

In agreement with Frakes and Fox [11], we find that programming language (or *Development approach*), experience, rewards, repository, and reuse measurement are not decisive factors, while reuse education (*Human factors*) is. Reuse measurement is not a variable in our data set, but interviews indicated that very few companies had a reuse measurement program in place. They report type of industry as a factor. In our study, we collected this information but did not use it in analysis, preferring other variables describing the type of software produced. Our rationale is that the type of industry is not sufficient to describe the type of software produced. For instance, a telecom company can produce MIS-like systems for customer management and billing, and embedded real-time systems for switching.

Size of company is not considered by Rine and Sonneman, or by Lee and Litecki. Frakes and Fox find it is not a factor. We also find that size is not a factor, in the sense that both small and large companies can succeed. However, it is a factor indirectly, as it influences the ease with which management commitment is obtained, and the reuse organization that can be sustained by the company.

7.2 Experience Reports from Companies

Hewlett-Packard was one of the early adopters of reuse. Griss [13] describes the general approach taken. Fafchamps [8] discusses organizational issues and choices. Lim [23] presents measured results from three HP divisions.

A summary of the lessons learnt can be found in [15]. We quote from this paper. "Technology is neither the major impediment to effective reuse nor the most critical success factor." "Objects and complex technology are not essential; complicated libraries are not necessary; performance concerns are largely a red herring; and process maturity is not as important as some would have you to believe." "Reuse is a business issue that involves technology transition and organizational change. Instituting a reuse culture, providing training, adhering to standard, and securing management commitment are the key success factors."

Our study confirms most of those findings. The variables that chiefly influence success are not technical. Two failures were related to the performance of embedded real-time software, but the root cause was not technical. We probably agree on process maturity, which we do not identify as a significant factor; since it was above average in almost all cases, however, we cannot wholly exclude it as a possible factor. Our study did not address the issue of standards.

Fafchamps [8] reports that the organization that worked best in HP was the "team producer" (equivalent to our *Independent team)* but three others were used, each having pros and cons. We find that different organizations are used, while the key issue is adaptation to the context.

Experimentation with reuse at IBM and Loral has been summarized by Poulin [29] and Tracz [34]. Poulin [31] describes specifically the lessons learnt about repositories, use of incentives, and the importance of domain-specific reuse and domain analysis to achieve higher levels of reuse. Our study confirms their results about the importance of management commitment, of small but effective repositories, and of the nonimportance of tools for classifying and searching assets.

Joos [18] describes reuse introduction efforts at Motorola. The paper highlights the highly challenging task of introducing reuse in a multinational corporation, already involved in other cultural changes, from hardware-centric to software-centric, and towards process maturity. The key issues recognized are acquiring commitment from top-level management, and providing reuse training and incentives.

Isoda [17] reports on a reuse project at NTT. He stresses the importance of the selection of a domain with high reuse potential, the need for domain analysis, and management. This is confirmed in our study, except that domain analysis is not identified as a key factor since several successful projects did not perform it.

Card [4] states that "the most important obstacles to reuse are economic and cultural, not technological." A reuse program is a technology transition problem and should be market-driven (reusable assets should be produced so that they are reusable by future projects). Cultural factors hinder technology transition and should be addressed by training, incentives, measurement, and management commitment. Our study confirms the essence of these statements, except for incentives and measurement. Few companies had a measurement program in place and even fewer used incentives.

Fichman and Kemerer [9] describe four longitudinal case studies in the early adoption of object-oriented techniques in the 1992-1996 period. Reuse was not achieved in any of the cases because of the misconception that "object-orientation = reuse." They argue that a company should decide on adopting an object-oriented approach or reuse, but not both at the same time. Our study confirms completely this finding.

Paci and Hallsteinsen [27] report on 15 reuse projects in European companies, most of them influenced by the REBOOT approach [21]. The book is an excellent source of information on economic results from long-term application of reuse and planned evolution and maintenance.

8 Validity

We discuss here possible objections to the validity of this empirical work. We use the definitions of construct, internal and external validity given by [19].

8.1 Construct Validity

Construct validity considers whether the metrics and models used in a study are a valid abstraction of the real world under study. In our case, this applies to the variables chosen to characterize the data set. Most of those variables are taken directly, or with little modification, from existing reuse models.

The dependent variable, success, or failure of a project is the most sensible one. As already mentioned, quantitative indicators such as return on investment, were rarely available. We defined success as a combination of objective (continuation of the reuse program, actual reuse of assets) and subjective assessments.

As will be detailed later, all projects received funding from the European Commission. Continuation of the reuse program (after the funding is over) is especially meant to filter any bias from external funding, as unsuccessful reuse programs are very unlikely to continue after funding is over.

The final value of this variable was further discussed, case by case, and before data analysis, between the three authors to avoid bias. In most cases, this was made easier because failures were openly admitted by the interviewees.

8.2 Internal Validity

Internal validity considers whether the experimental design is able to support conclusions on causality or correlations. The size of our data set is too limited to allow meaningful statistical studies, so our study is exploratory.

On the other hand, our data set is, as far as we know, the biggest (and the only one) available on industrial reuse projects. Further, it is based on direct interviews with participants in the projects, what makes it possible to gain much better knowledge of the projects as compared with mailed surveys. In comparison with experience reports from companies, our interviews reach a lower level of detail, but consider a variety of companies, including small and very small ones.

8.3 External Validity

The companies in the data set are all located in Europe. Nevertheless, we cannot see any influence of this geographic factor on the attributes that we investigated, either in terms of the state and or the control variables. Therefore, we do not think that the validity of the results of this study could be influenced by this factor.

There is no discrimination between the companies in terms of size. In the sample, many small and very small companies, as well as large ones, are represented.

On the other hand, all the companies interviewed won funding from the European Commission to perform a technology transfer project. The competition to win funding is demanding. A detailed proposal has to be written. The proposals are evaluated and ranked by a panel of senior industrial and academic experts on technical and management criteria. The acceptance ratio is typically 1 to 10. For these reasons, we believe that the sample of companies does not represent the average European companies, but are probably above average, as far as technical and managerial skills are concerned. This factor should definitely be kept in mind when considering external validity. In the data set, this factor is indirectly confirmed by the above-average level of process maturity (*Software process maturity* variable).

9 Conclusion

We have presented the results of a survey of European companies involved in introducing and implementing reuse programs.

Projects were performed in large and small companies, working in a variety of business domains. Most of them produce software with high commonality between applications, have a good process maturity level, and use an object-oriented or procedural development approach. Despite this potential for success, around one-third of projects failed.

Failures were due to not introducing reuse processes, not modifying nonreuse processes, and not considering human factors. The root cause was the lack of commitment by top management, or nonawareness of the importance of these factors, often coupled with the belief that using the object-oriented approach or setting up a repository would automatically lead to success in reuse.

Given a reuse potential due to commonality among applications, the success of a reuse initiative depends on a mix of features.

1. Overall, initiating and succeeding in a reuse initiative is a technology transfer endeavor, which requires, as a sine qua non, commitment of management.
2. The approach to designing a reuse program seems to be standard, or at least requires considering the same set of elements. Initiating reuse processes, modifying nonreuse processes, and addressing human factors.
3. If the approach is standard, the way of deploying it is not. Each element listed above must be approached according to the context of the company.

We have modeled these success factors with a decision sequence.

These results are in conformance with most of the studies already performed. However, this is the first study, to our knowledge, to use interviews and, therefore, to have a higher degree of confidence in the results. Further, small companies were also involved, in a variety of countries and cultures.

To advance the state of practice, the results of this study should be used to eradicate the misunderstandings that are still popular among practitioners. Unfortunately, reuse is seldom part of software engineering curricula, so students do not contribute to advancing the state of knowledge when they are hired in companies. Overall, it is easy to forecast there will be a long delay before sensible advances are made.

From the research point of view, we have remarked on the difficulty of characterizing reuse in companies using the conceptual tools currently available. Reuse, for example in the definition used for this study, is an umbrella concept capable of encompassing a variety of situations. If we want more analytic power, we need better analytic tools to understand the various instantiations of reuse while keeping a common high-level definition.

Further, we observe a shift from reusable assets based on code, to reusable components, acquired or developed internally. Consequently, research on reuse should focus more on the implications of the component paradigm on processes, organizations, and tools for reuse.

Appendix

Formal Analysis of the Data Set

This appendix contains the details of a formal analysis of the data set, comprising a classification tree analysis of successes and failures, and a cluster analysis limited to successes.

A.1 Successes and Failures

We look for a correlation between independent variables (restricted to state variables and high-level control variables, see Tables 1 and 2) and the dependent variable (success or failure). We consider all cases, except X, which failed for reasons outside the scope of reuse (project X is analyzed in the next section).

We use classification trees [2], [28] for the analysis because they handle variables with nominal and ordinal scales, they provide results easy to interpret, and they can handle data sets with few cases. Classification trees are used to predict membership of cases in the classes of a nominal dependent variable from their values on one or more predictor variables. Predictor variables can be nominal or ordinal. A tree is built using a training set and its predictive accuracy is evaluated against a test set.

A tree building algorithm selects an independent variable (e.g., human factors), selects a condition to split the data set (e.g., human Factors = "no"), and evaluates the subsets obtained with the split to decide on changing the split condition or not. The procedure is recursively applied to each subset. Splitting is stopped when a subset contains only cases belonging to one class, or when a minimum number of cases is reached for a subset.

We report in Fig. 6 the classification tree obtained using the CART algorithm, with the Gini measure of node impurity as the splitting criterion, and FACT style direct stopping [2]. CART searches exhaustively the best split condition among all the possible ones in a certain node of the tree. The Gini measure evaluates a splitting criterion by favoring homogeneous subsets. FACT style stops when a subset contains only cases belonging to one class. In the figure, the splitting condition is plotted under a node. The number on an arc is the number of cases in the subset defined by the split. The number on the left top corner of a node is the identifier of a node. The label on the top right corner of each node represents the class of the node as concerns the dependent variable (for instance node 1 is tagged with success = "yes"). The tree classifies correctly all the cases in the training set.

Fig. 7 reports the corresponding importance ranking for predictor variables. When the tree is built, each available variable is considered as a candidate for the next split. As one variable is chosen, surrogate variables are also considered and ranked. The ranking of surrogate variables is aggregated for all nodes and its value is plotted, scaled

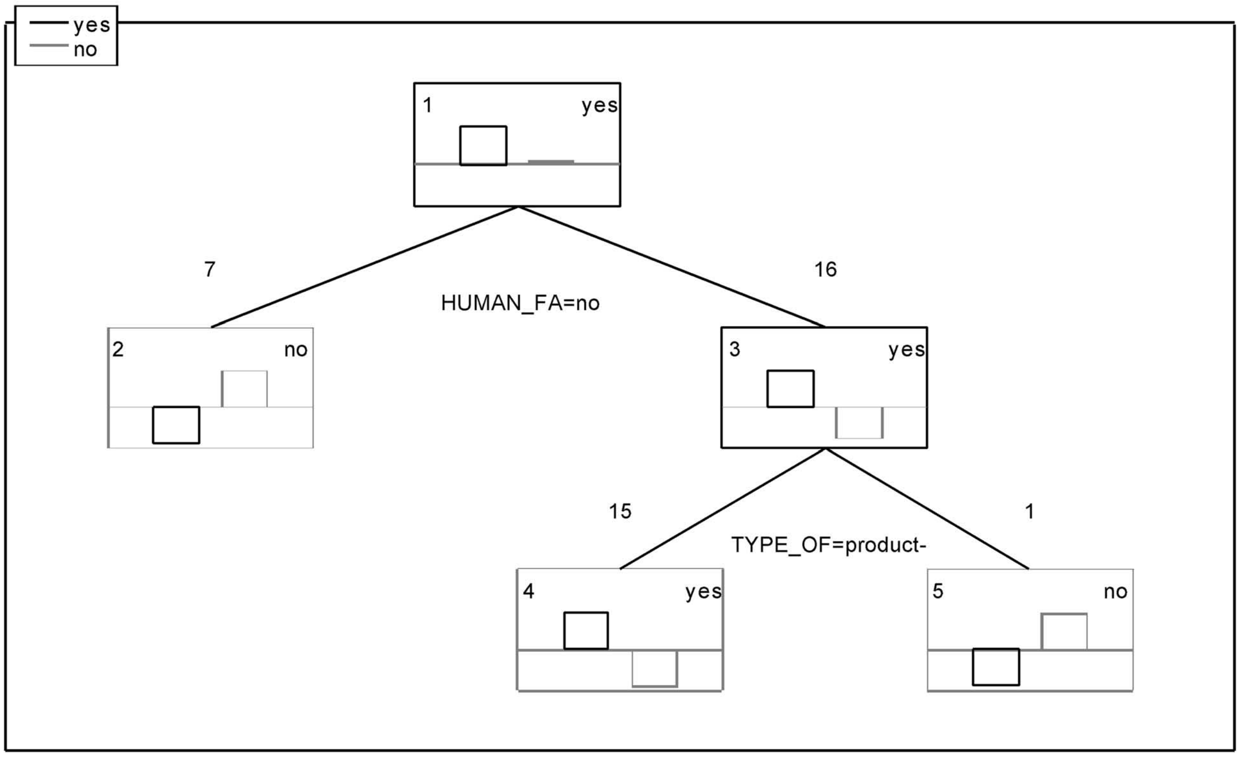


Fig. 6. Classification tree for state and high-level control variables.

relative to the best predictor variable. The value is a measure of variable importance [2].

Given the high number of variables and the low number of cases, we had to exclude two variables from the analysis. We chose *Application domain* and *Size of baseline*. Application domain gives insight into the business context, but *Type of software production*, *Software and product*, and *Type of software* thoroughly describe the technical context. We deemed *Size of baseline* not essential for the analysis since it is a low-level characteristic of a project that we collected mainly to exclude toy applications of reuse. *Repository* is also excluded since it has value "yes" for all cases.

In the tree, the variables *Human factors* and *Type of software production* are able to correctly classify all the cases. The variable ranking (Fig. 7) gives a more comprehensive picture, reporting also *Top management commitment*, *Reuse processes introduced*, and *Nonreuse processes modified*.

Using different classification algorithms, the tree and the ranking change slightly. However, a result is constant. The variables *Type of software production*, *Top management commitment*, *Reuse processes introduced, Nonreuse processes*

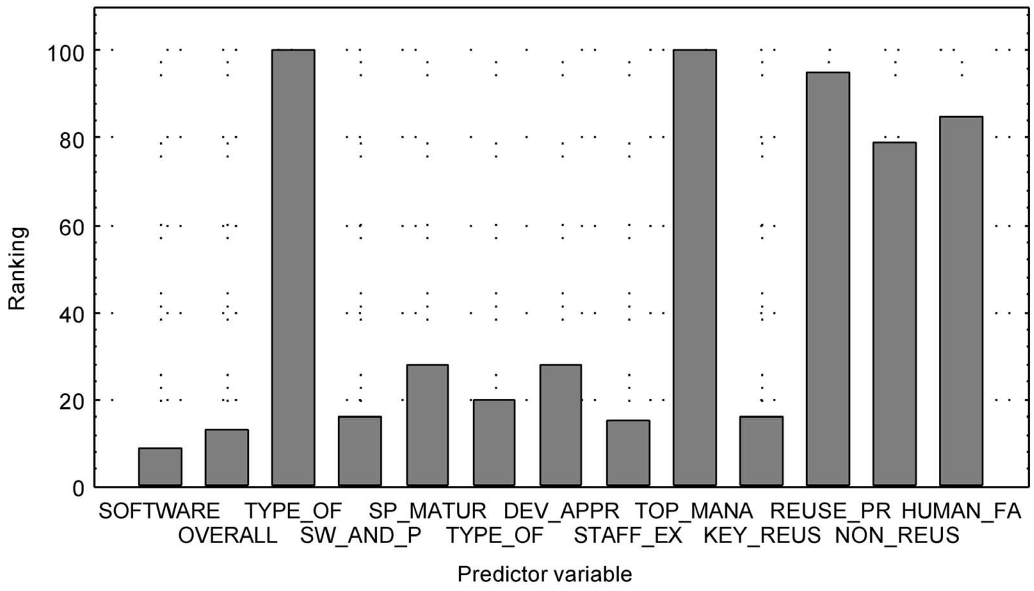


Fig. 7. State and high-level control variables predictor importance ranking.

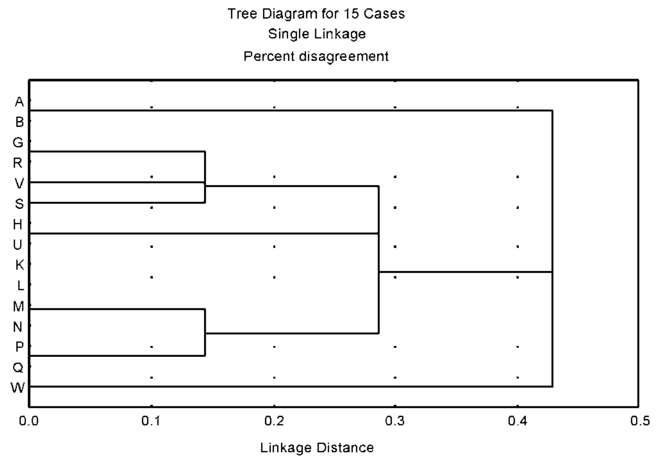


Fig. 8. Low-level control variables, cluster analysis.

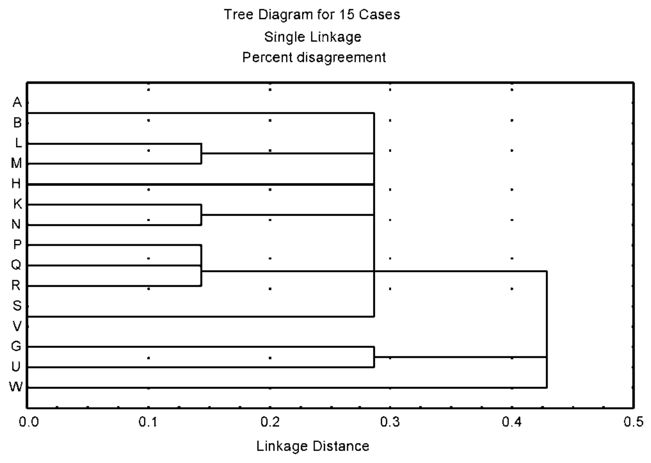


Fig. 9. State variables, cluster analysis.

modified, and *Human factors* have in all cases more predictive importance than the rest.

In other words, of the state variables only *Type of software production* has predictive importance; of the high-level control variables, all have predictive importance except *Repository* (excluded from analysis since always "yes") and *Key reuse roles introduced*.

A.2 Successes, State, and Low-Level Control Variables

The analysis of failures and successes has shown that successful projects address in substantially the same way a set of issues, identified by the high-level control variables. But, at a lower level of detail, as expressed by low-level control variables, successful projects behaved differently. So, now we look for recurring patterns at this level.

First, we analyze low-level control variables, then we analyze state variables. Ideally, we would like to find patterns in low-level control variables, patterns in state variables, and a correlation between them.

We use cluster analysis [7] with the goal of grouping cases in clusters that addressed reuse issues in similar ways. Cluster analysis is a useful tool in exploratory data analysis.

Cluster analysis proceeds recursively, merging similar cases in groups, then similar groups in clusters. A measure of distance is used to evaluate similarity. We use percent disagreement as a measure of distance between cases. Percent disagreement can be used on nominal scales and, computes, for a pair of cases, the normalized number of variables with different values.

As a measure of distance between groups, we use single linkage, the distance between the closest pair of cases in the groups. Figs. 8 and 9 show the resulting clusters. Cases are on the Y axis and the X axis shows the distance value at which cases are merged.

Fig. 8 contains the results of cluster analysis on low-level control variables. Clustering considers all low-level control variables, except *Rewards* (= no for all cases), *When assets developed* (dependent on *Independent team)*, and *Number of assets* (projects did not set goals on the number of assets to be developed, so this variable is more a dependent variable than a control).

Projects A and B are in a cluster apart. This depends on the values of *Origin*, *Independent team*, and *When developed*.

Project W is a case apart, but this is mainly an effect of the clustering algorithm. The difference is due to the "no" on *Qualification* and *Configuration management*. If we do not consider these variables, it clusters with G, R, S, and V.

The other cluster contains all cases except A and B. Their common characteristic is that they do not use the more sophisticated approach of A and B. Assets are produced by projects, just in time, in most cases, by reengineering legacy code. G, R, S, V, and W produce only code assets. K, L, M, N, P, and Q produce also requirements and design assets.

In summary, the cluster analysis on low-level control variables divides cases according to the organization to produce assets as discriminated by *Origin*, *Independent team*, and *When developed*.

We now analyze the state variables (Fig. 9).

The two main clusters (G, U, and W, and the rest) are discriminated by the value of *Type of software* (= Business for G, U, and W). This clustering is very sensitive to the linkage distance used. Using other distance measures, clustering varies.

Overall, cluster analysis on state variables does not provide reliable clusters. On low-level control variables, two clusters are identified, A and B, and the rest. No further meaningful grouping can be found.

Acknowledgments

This study has been possible thanks to the support and collaboration of several people. The authors would like to thank the European Commission for funding the Surprise ESPRIT/ESSI project number 23460 and, specifically, project officers Andrea Di Maio and Gisèle Roesems; project managers and staff of the PIEs for providing their time and suggestions during the interviews; and Carlo Ghezzi for his initial encouragement. They are also grateful to Vic Basili, Davide Brugali, Reidar Conradi, and Sandro Morasca for reviewing the manuscript and providing constructive comments. Daniele Romano gave precious hints for the analysis of the data set. During the data analysis phase, Maurizio Morisio was visiting the Experimental Software Engineering Group in the Institute for Advanced Computer Studies (Umiacs) at the University of Maryland, College Park, on sabbatical leave from Politecnico di Torino.

REFERENCES

[1] J. Bayer, O. Flege, and P. Knauber, "PuLSE: A Methodology to Develop Software Product Lines," *Proc. Symp. Software Reusability (SSR '99)*, May 1999.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Wadsworth, 1984.

[3] B.W. Boehm, "Economic Analysis of Software Technology Investments," *Analytical Methods in Software Eng. Economics,* T. Gulledge and W. Hutzler eds., 1993.

[4] D. Card, "Why Do So Many Reuse Programs Fail?" *IEEE Software,* pp. 114-115, Sept. 1994.

[5] ESI, The Vasie Project, 1998. Available at http://www.esi.es/VASIE.

[6] ESSI Software Best Practice, Summaries of ESSI Projects, *European Commission Directorate General XIII,* 1997. Available at http://www.cordis.lu/esprit/src/projects.htm.

[7] B. Everitt, *Cluster Analysis,* third ed. John Wiley, 1993.

[8] D. Fafchamps, "Organizational Factors and Reuse," *IEEE Software,* pp. 31-41, Sept. 1994.

[9] R.G. Fichman and C.K. Kemerer, "Object Technology and Reuse: Lessons from the Early Adopters," *Computer,* vol. 30, no. 10, pp. 47-59, Oct. 1997.

[10] W.B. Frakes and C.J. Fox, "Sixteen Questions about Software Reuse," *Comm. ACM,* vol. 38, no. 6, June 1995.

[11] W.B. Frakes and C.J. Fox, "Quality Improvement Using a Software Reuse Failures Model," *IEEE Trans. Software Eng.,* vol. 23, no. 4, pp. 274-279, Apr. 1996.

[12] W.B. Frakes and S. Isoda, "Success Factors of Systematic Reuse," *IEEE Software,* pp. 14-19, Sept. 1994.

[13] M.L. Griss, "Software Reuse: From Library to Factory," *IBM System J.,* vol. 32, no. 4, 1993.

[14] M.L. Griss and M. Wosser, "Making Reuse Work at Hewlett-Packard," *IEEE Software,* pp. 105-107, Jan. 1995.

[15] M.L. Griss, "Software Reuse: Objects and Frameworks are not Enough," *Object Magazine,* pp. 77-87, Feb. 1995.

[16] M.L. Griss, P. Jonsson, and I. Jacobson, *Software Reuse.* Addison-Wesley, 1997.

[17] S. Isoda, "Experiences of a Software Reuse Project," *J. System and Software,* vol. 30, no. 3, pp. 171-186, Sept. 1995.

[18] R. Joos, "Software Reuse at Motorola," *IEEE Software,* pp. 42-47, Sept. 1994.

[19] C.M. Judd, E.R. Smith, and L.H. Kidder, *Research Methods in Social Relations,* sixth ed. Holt Rinehart and Winston, 1991.

[20] B. Kain, "Pragmatics of Reuse in the Enterprise," *Object Magazine,* pp. 55-58, Feb. 1994.

[21] E.A. Karlsson, *Software Reuse.* John Wiley & Sons, 1995.

[22] C. Kruger, "Software Reuse," *ACM Computing Surveys,* vol. 24, no. 2, pp. 131-183, 1992.

[23] W.C. Lim, "Effects of Reuse on Quality, Productivity and Economics," *IEEE Software,* pp. 23-30, Sept. 1994.

[24] N.Y. Lee and C.R. Litecky, "An Empirical Study of Software Reuse with Special Attention to Ada," *IEEE Trans. Software Eng.,* vol. 23, no. 9, pp. 537-549, Sept. 1997.

[25] H. Mili, F. Mili, and F.A. Mili, "Reusing Software: Issues and Research Directions," *IEEE Trans. Software Eng.,* vol. 21, no. 6, pp. 528-561, June 1995.

[26] M. Morisio, C. Tully, and M. Ezran, "Diversity in Reuse Processes," *IEEE Software,* pp. 56-63, July/Aug. 2000.

[27] M. Paci and S. Hallsteinsen, *Experiences in Software Evolution and Reuse,* 1997.

[28] A. Porter and R.W. Selby, "Empirically Guided Software Development Using Metric-Based Classification Trees," *IEEE Software,* pp. 46-54, Mar. 1990.

[29] J. Poulin, "Reuse: Been There, Done That," *Comm. ACM,* vol. 42, no. 5, pp. 98-100, May 1999.

[30] J. Poulin, *Measuring Software Reuse.* Addison Wesley, 1996.

[31] J. Poulin, "Populating Software Repositories: Incentives and Domain-Specific Software," *J. System and Software,* vol. 30, no. 3, pp. 187-199, Sept. 1995.

[32] D.C. Rine and R.M. Sonneman, "Investments in Reusable Software: A Study of Software Reuse Investment Success Factors," *J. Systems and Software,* vol. 41, pp 17-32, 1998.

[33] B.C. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Trans. Software Eng.,* vol. 25, no. 4, pp. 557-572, July 1999.

[34] W. Tracz, "Confessions of a Used-Program Salesman: Lessons Learnt," *Proc. Symp. Software Reliability, (SSR 95),* pp. 11-13, 1995.

[35] D.M. Weiss and C.T.R. Lai, *Software Product-Line Engineering: A Family-Based Software Development Approach.* Addison-Wesley, 1999.

Maurizio Morisio received the MSc degree in electronic engineering and the PhD degree in software engineering from Politecnico di Torino, Italy. He is a research assistant in the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy. He recently spent two years working with the Experimental Software Engineering Group at the University of Maryland, College Park. During that time he was codirector of the Software Engineering Laboratory (SEL), a consortium of NASA Goddard Space Flight Center, the University of Maryland, and Computer Science Corporation, which has the mission of improving software practices at NASA and CSC. The overall goal of Maurizio's research and consulting is to understand how software is produced and maintained, in order to improve software processes and products in industrial settings. Software production involves three main dimensions (processes, people and organization, tools and technology) and his activity has spanned all three, including work on object-oriented technology (analysis, design, and programming), software product lines, framework-based development, COTS-based development, processes and measures for individuals and small teams (PSP, PIPSI), and the evaluation and selection of tools. His current focus is on open source development and service engineering for the wireless internet. His approach to both research and consulting is strongly empirical: observing and analysing facts rather than trusting claims and hype, and using empirical methods such as case studies, experiments, and surveys. He is a member of the IEEE Computer Society.

Michel Ezran is chief knowledge officer at the Paris office of Valtech, a leading international e-business consulting firm operating in eight countries across Europe, North America, and Asia. As the leader of Valtech's corporate knowledge management program, Michel manages a team of knowledge managers, a worldwide network of experts, and Valtech's enterprise portal. He joined Valtech as staff number 10, and has successively held positions of senior consultant and research and development manager. As a consultant, he advised companies in migrating their business information systems to new technologies (Java, CORBA, UML, EJB, internet), including not only the technology aspects but also architecture, process, organization, and management, and helped software organizations improve their processes through the adoption of reuse and business components. He has also acted as a consultant to advanced technology projects, on the coordination of development teams, on monitoring development phases (requirements capture, object analysis, architecture definition, object design, coding, and testing), and on managing the risks arising from migration to new technologies. Prior to Valtech, he worked for Cap Gemini in France and South America, and for other software houses, with experience on a variety of software development projects in the field of business information systems.

Colin Tully is a professor of business information systems in the School of Computing Science at Middlesex University, London, UK. The School is oriented strongly toward the applied end of the discipline. Its strengths lie in areas such as human-computer interaction modeling and design, usability, systems failures, systems and software processes and their improvement, development methods, neural networks, vision and image processing, multimedia programming, hypermedia authoring, digital libraries, mobile and personal technologies, medical informatics, and telematics. He is a member of the IEEE.