Empir Software Eng (2007) 12:471–516
DOI 10.1007/s10664-007-9040-x

Quality, productivity and economic benefits of software reuse: a review of industrial studies

Parastoo Mohagheghi · Reidar Conradi

Published online: 3 May 2007

Editor: Katsuro Inoue

Abstract Systematic software reuse is proposed to increase productivity and software quality and lead to economic benefits. Reports of successful software reuse programs in industry have been published. However, there has been little effort to organize the evidence systematically and appraise it. This review aims to assess the effects of software reuse in industrial contexts. Journals and major conferences between 1994 and 2005 were searched to find observational studies and experiments conducted in industry, returning eleven papers of observational type. Systematic software reuse is significantly related to lower problem (defect, fault or error) density in five studies and to decreased effort spent on correcting problems in three studies. The review found evidence for significant gains in apparent productivity in three studies. Other significant benefits of software reuse were reported in single studies or the results were inconsistent. Evidence from industry is sparse and combining results was done by vote-counting. Researchers should pay more attention to using comparable metrics, performing longitudinal studies, and explaining the results and impact on industry. For industry, evaluating reuse of COTS or OSS components, integrating reuse activities in software processes, better data collection and evaluating return on investment are major challenges.

Keywords Software reuse · Review · Quality · Productivity · Evidence

P. Mohagheghi (✉)
SINTEF, ICT, P.O.BOX 124 Blindern, 0314 Oslo, Norway
e-mail: parastoo.mohagheghi@sintef.no

P. Mohagheghi · R. Conradi
Department of Computer and Information Science, Norwegian University of Science and Technology, 7491 Trondheim, Norway

R. Conradi
e-mail: conradi@idi.ntnu.no

1 Introduction

There is extensive literature on systematic software reuse (or in-short reuse); its purpose and promises, how to develop for and with reuse, technical/managerial/organizational aspects, measuring reuse rate and Return-On-Investment (ROI), and success and failures of reuse practices. However, little research has accumulated reported results in a review type paper like this, where the goal is collecting and appraising evidence and finding research gaps for future studies.

In this paper, we summarize empirical quantitative evidence from industry on reuse appeared between 1994 and 2005. The question guiding the review is: "To what extent do we have evidence that software reuse leads to significant quality, productivity or economic benefits in industry?." Specifically the aim of this review is to (1) find and organize the quantitative empirical evidence from industry related to the review question, (2) evaluate the quality of reporting, identify metrics, data collection procedures and analysis methods, (3) summarize the findings, and (4) identify gaps for future research. The review defines five research questions that are answered through the evidence. The possible audience of this review are two groups: those who plan future studies on reuse may learn from experience to improve the state of research, and those who seek evidence on reuse benefits for decision-making may use this review as a reference.

The remainder of the paper is structured as follows. Section 2 provides definitions of concepts used in the review. Section 3 presents the review process in terms of research questions, the framework for performing the review, paper selection criteria and validity threats. Section 4 gives an overview of the reviewed papers and Sections 5–7 asses the papers regarding metrics used, data collection and analysis, and summarizes findings. Section 8 discusses shortcomings in reuse research and ideas for future research and Section 9 presents lessons for improving research. The review is concluded in Section 10.

2 Concepts

There is a diversity of definitions in literature on reuse and types of studies, and our purpose here is not to review the definitions, but to present what we mean when using these terms.

2.1 Software Reuse

Software reuse is the systematic use of existing software assets to construct new or modified ones or products. Software assets in this view may be source code or executables, design templates, free standing Commercial-Off-The-Shelf (COTS) or Open Source Software (OSS) components, or entire software architectures and their components forming a product line or product family. Knowledge may also be reused and knowledge reuse is partly reflected in the reuse of architectures, templates or processes. Developing components so that they become reusable is called developing *for reuse*, while developing systems of reusable components is called developing *with reuse* (Karlsson 1995). Both these are covered in the review. *Reusability* is a property of a software asset that indicates its probability of reuse (Frakes and Kang 2005). *Ad-hoc* reuse in this review means that reuse is opportunistic and not part of a repeatable process, as opposed to *systematic reuse*; meaning planned. Glass (2002) discusses that "ad hoc" originally means "for this" or

"suited for the task at hand," and is different from not planned or not repeatable. However, the term is widely used in other literature as opposed to systematic reuse, and this review uses it as well.

Almost any software today is built on software developed by others; for example operating systems, programming languages' libraries, CASE tools, debuggers, desktop applications, databases or application servers. Reuse in this review does not cover reuse of the above software which is not considered to be developed by the company itself, but would be purchased or obtained as OSS products for software development.

Sometimes reuse refers to developing new releases of assets or products based on the previous releases (Basili 1990). We call this release-based or incremental development, which is in fact a maintenance and evolution activity. Frakes and Terry (1996) call this for "carry-over reuse." This type of reuse is not included in this review.

2.2 Study Types

We decided to include all studies reporting quantitative results from industry related to reuse in the review and then classify the study type, leaving out surveys and papers with discussion but no hard data. The study type is important information in each study since it communicates what is expected from a study and how the evidence should be evaluated. However, a search of literature for study types showed that there are not consistent definitions and/or the definitions are not communicated well. Therefore, we have to define our perspective of study types.

One definition of study types that is applied on empirical research is given by Zannier et al. (2006) (see the paper for a complete list of their references). Table 1 also shows these definitions and some other that we found.

Zannier et al. (2006) analyzed a random sample of 63 papers published in 29 ICSE (the International Conference on Software Engineering) proceedings since 1975 using the above classification. Authors of only 25 papers had defined their study type, and Zanneir et al. give both authors and their perspective of the study types. We use their definitions but also add that when studies are performed at a single point in time, they are called *cross-sectional*, as opposed to *longitudinal studies*.

A case study may be comparative, and Kitchenham and Pickard (1998) describe three methods of comparison in a quantitative case study, which are (a) comparing the results of using a new method with a *company baseline*, (b) comparing components within a project that are randomly exposed to a new method to others or *within project component comparison*, and (c) comparing a project using a new method to a sister project that uses the current method or *sister-project case studies*. An alternative sister-project design is developing a product twice using different methods or *replicated product design*. This review found examples of (b) and (c) in different types of studies, and we hence call the method of comparison for *component-comparison* (components may be from one or several products) and *sister-project comparison* (including *replicated product design*).

2.3 Objects and Subjects of Study, Variables and Measurement

We use the definitions of variables, treatments, objects and subjects of study of Wohlin et al. (2000). The *object of study* is the entity that is studied; for example, a program that shall be developed with different techniques. The people that apply the treatment are called *subjects*, for example the developers of a software product. The characteristics of both the objects and the subjects can be independent variables in a study. All variables that are manipulated

Table 1 Study types and their definitions

Study type	Definition as given in Zannier et al. (2006)	Other definitions
Controlled experiment	Random assignment of treatment to subjects, large sample size (>10), well-formulated hypotheses and independent variable selected. Random sampling.	*Controlled study* (Zelkowitz and Wallace 1998). *Experimental study* where particularly allocation of subjects to treatments are under the control of the investigator (Kitchenham 2004). *Experiment* with control and treatment groups and random assignment of subjects to the groups, and *single-subject design* with observations of a single subject. The randomization applies on the allocation of the objects, subjects and in which order the tests are performed (Wohlin et al. 2000). *Experiments* explore the effects of things that can be manipulated. In randomized experiments, treatments are assigned to experimental units by chance (Shadish et al. 2001). Our note: Randomization is used to assure a *valid sample* that is a representative subset of the study population; either in an experiment or other types of study. However, defining the study population and a sampling approach that assure representativeness is not an easy task, as discussed by Conradi et al. (2005).
Quasi-experiment	One or more points in Controlled Experiment are missing.	In a *quasi-experiment*, there is a lack of randomization of either subjects or objects (Wohlin et al. 2000). *Quasi-experiment* where strict experimental control and randomization of treatment conditions are not possible. This is typical in industrial settings (Frakes and Succi 2001). *Quasi-experiments* lack random assignment. The researcher has to enumerate alternative explanations one by one, decide which are plausible, and then use logic, design, and measurement to assess whether each one is operating in a way that might explain any observed effect (Shadish et al. 2001).
Case study	All of the following exist: research questions, propositions (hypotheses), units of analysis, logic linking the data to the propositions and criteria for interpreting the findings (Yin 2003).	A *case study* is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly

Table 1 (Continued)

Study type	Definition as given in Zannier et al. (2006)	Other definitions
		evident. A *sister-project case study* refers to comparing two almost similar projects in the same company, one with and the other without the treatment (Yin 2003). *Observational studies* are either case studies or field studies. The difference is that multiple projects are monitored in a field study, may be with less depth, while case studies focus on a single project (Zelkowitz and Wallace 1998). Case studies fall under *observational studies* with uncontrolled exposure to treatments, and may involve a control group or not, or being done at one time or historical (Kitchenham 2004).
Exploratory case study	One or more points in case study are missing.	The propositions are not stated but other components should be present (Yin 2003).
Experience report	Retrospective, no propositions (generally), does not necessarily answer why and how, often includes lessons learned.	Postmortem Analysis (PMA) for situations such as completion of large projects, learning from success, or recovering from failure (Birk et al. 2002).
Meta-analysis	Study incorporates results from several previous similar studies in the analysis.	*Historical studies* examine completed projects or previously published studies (Zelkowitz and Wallace 1998). Our note: Meta-analysis covers a range of techniques for summarizing findings of studies.
Example application	Authors describe an application and provide an example to assist the description. An example is not a type of validation or evaluation.	Our note: If an example is used to evaluate a technique already developed or apply a technique in a new setting, it is not classified under example application.
Survey	Structured or unstructured questions given to participants.	The primary means of gathering qualitative or quantitative data in surveys are interviews or questionnaires (Wohlin et al. 2000). *Structured interviews* (qualitative surveys) with an interview guide, to investigate rather open and qualitative research questions with some generalization potential. *Quantitative surveys* with a questionnaire, containing mostly closed questions. Typical ways to fill in a questionnaire are by paper copy via post or possibly fax, by phone or site interviews, and recently by email or web (Conradi et al. 2005).
Discussion	Provided some qualitative, textual, opinion- oriented evaluation.	Expert opinion (Kitchenham 2004).

and controlled are called *independent variables*. Those variables that we want to see the effect of the changes in the independent variables are called *dependent variables*. A *treatment* is one particular value of an independent variable. The treatments are being applied to the combination of objects and subjects. A *confounding factor* is a factor that makes it impossible to distinguish the effects from two treatments from each other, such as different skills of developers.

Measurement is here used for the activity of measuring a property of software, a *metric* is the property of software that is measured; for example software size in Lines of Code (LOC), while a *measure* refers to the symbol or number that is assigned to the property by the activity of measurement.

For case studies, Yin (2003) recommends defining *unit of analysis* or what the "case" is; for example individuals, a product or an organization, and *sources of evidence* which may be documentation, archival records, interviews, direct observations, participant-observation, and physical artifacts. Using multiple sources of evidence is strongly recommended to increase reliability of a case study. Since this terminology is not used in the papers, we have not summarized papers regarding their sources of evidence.

3 The Review Process

This section presents the review framework and the five research questions that are derived from the review question, the paper inclusion criteria and the validity threats of the review.

3.1 Review Framework and Research Questions

In this review, we ask "To what extent do we have evidence that software reuse leads to significant quality, productivity or economic benefits in industry?". This research was initiated related to a study we performed on quality benefits of reuse. However, in spite of our expectation for overwhelming evidence, the search for papers showed that reported results from industry are surprisingly few. In addition to the sparseness of the results, the question of practical significance is rarely discussed. We also searched for experiments in artificial settings, which only added one student experiment to the search results (Basili et al. 1996) that is not included in this review.

The formulation of the review question follows recommendations by Dybå et al. (2005) for collecting evidence as answer to questions. Questions should be well-partitioned into *intervention*, *context* and *effect*. In this review, the intervention is "software reuse," the context is "industrial settings" and the effect is "changes in quality, productivity or Return-On-Investment (ROI)." The intervention is either directly or indirectly measured in reuse metrics, while the effect is measured in dependent variables such as problem density. Figure 1, inspired from Wohlin et al. (2000) and Dedrick et al. (2003), shows the framework leading this review.

We have also added the appraising view to the question by asking the *significance* of the results. Specifically we ask the following research questions:

RQ1 What types of studies are performed and what data are reported on the reuse approaches?
RQ2 Which metrics are used for reuse and its effects?
RQ3 How are quantitative data reported and analyzed?
RQ4 What are the findings and what theory may be developed based on the findings?
RQ5 What are the shortcomings regarding reuse research?

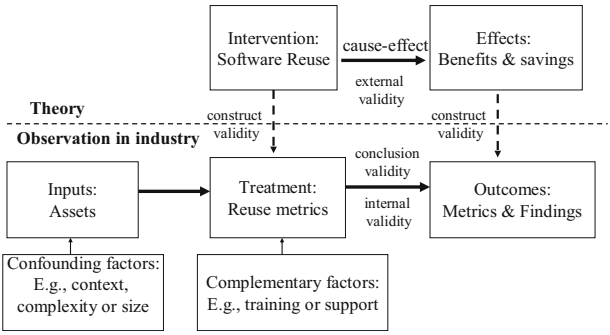

Fig. 1 The review framework

RQ1 to *RQ5* are discussed in Sections 4–8 successively. Since we have not found any such review of earlier research, we need to perform a detailed exploratory analysis of papers. Our guide in performing this review has specially been: Webster and Watson (2002), Kitchenham (2004), Kitchenham et al. (2002), Gregor (2002) and Pickard et al. (1998).

3.2 Paper Inclusion Criteria

The review concentrates on studies whose results are published in peer-reviewed journals and conferences. Additional sources would be books and technical reports (for example, Hallsteinsen and Paci 1997) that are not included in the review.

We searched the *ACM digital library* and the *IEEE Xplore* which also include many conference proceedings, *Empirical Software Engineering Journal, Journal of Systems and Software, Journal of Information Science, MIS Quarterly (MISQ)* from September 1994 (online), *IEEE Transactions of Software Engineering* (TSE), *IT Professional*, *ACM Computing Surveys (CSUR)*, and the *Journal of Research and Practice in Information Technology* (from 2003 online). We searched the above sources with keywords "reuse," "reuse benefits" and "reuse case study." To assure better coverage, proceedings of the *International Conference on Software Reuse (ICSR)*, the *IEEE International Conference on Software Maintenance* (ICSM) since 1995 online, the *International Software Product Line Conference (SPLC)* started in 2001, the *International Conference on Software Engineering (ICSE)* since 1995 online, the *International Conference on COTS-Based Software Systems (ICCBSS)* started in 2002, *MISQ* and the *IEEE Software* magazine were manually checked. We searched for papers reporting quantitative results but will discuss their qualitative findings as well. We searched for case studies and experiments but excluded surveys.

We only reviewed papers published from 1994 to 2005. Frakes and Terry (1996) provide a survey of reuse metrics and models from earlier research, and Hallsteinsen and Paci (1997) summarize some earlier research. This review differs from the above sources with respect to giving an explicit selection criterion and the searched resources, appraising evidence and discussing significance, classifying studies and covering new research.

The review process identified eleven papers that match our selection criterion, all retrieved in full text, and which compared systematic reuse with ad-hoc or no reuse, or compared reused components with the non-reused ones. In addition to these papers, Ramachandran and Fleischer (1996) included data on reuse rate but no quantitative findings on benefits and is therefore not included in the review. We also found three papers on reuse

of OSS components that are not included in the review either because they describe reuse of software for infrastructure or they lack quantitative data, or both reasons apply:

- Madanmohan and Dé (2004) performed structured interviews with developers of some commercial firms to find how they use OSS software. They classified the products as being operating systems, middleware, databases and support software. The paper has no data on ROI or quality.
- Norris (2004) writes that using OSS software for developing mission-critical software at NASA has reduced in-house effort and provided software with fewer bugs, without giving quantitative data.
- Fitzgerald and Kenny (2004) report on cost savings using OSS software when developing an infrastructure system for a hospital. Phase 1 of the project covered generic products such as an email system, a content management system and desktop applications and showed significant savings. Phase 2 would cover more specific products but at the time of publication, Phase 2 was still under planning and the savings were so far only estimated in the paper.

The final list therefore includes the following eleven papers ordered here after the year of publication: Lim (1994), Thomas et al. (1997), Frakes and Succi (2001), Succi et al. (2001), Morisio et al. (2002), Tomer et al. (2004), Mohagheghi et al. (2004),[1] Baldassarre et al. (2005), Morad and Kuflik (2005), Selby (2005), and Zhang and Jarzabek (2005).

3.3 Threats to the Validity of the Review

The main threats to validity of the review are:

- *Uncovered publication channels for external validity*: We chose the journals, conferences and libraries that in our experience publish major research results on software reuse. Additional search may add new papers, which needed more effort. Giving the inclusion criterion and the publication channels allows for validation and extension of the review.
- *Undetected papers for external validity*: We searched with a few keywords but to improve the detection process, we manually checked several publication channels. From the reviewed papers, only one of them does not include the word "reuse" in the title and was detected by manual check; i.e. (Morisio et al. 2002), which is an indication that we may have missed some papers but the extent is limited.
- *Publication bias for internal validity*: Probably success cases of reuse are published more often than failures, and significant results may be published more often than when the results are not considered as significant.
- *Researcher bias for construct validity*: Both authors have experience with industrial software reuse. We compared papers to determine relevant classifications and searched literature for definitions. The classifications and conclusions reflect our knowledge and opinion. The researchers have done their best to provide an objective review when analyzing research and we have presented all the results in the review to allow discussion and future extension. The main analysis is performed by the first author and the results are discussed with the second author.

[1] This paper is extended in Mohagheghi and Conradi (2007), and some information is taken from the extended version.

4 Answering *RQ1*—What Types of Studies are Performed and What Data are Reported on the Reuse Approaches?

In this section, we review the studies regarding the object of study, study type, domain, scale, publication channel, the year of publication, and the data reported on the reuse approaches.

4.1 Objects, Types of Studies, Scale, Publication Channel and Year

Appendix A gives an overview of the eleven reviewed papers, ordered after the year of publication. It also shows objects and type of studies. We applied the classification discussed in Section 2.2 while most papers also define the study type. The field "Agreement" shows whether there is agreement between the review's and authors' perspective of the study type. KLOC stands for Kilo Lines of source Code as a measure of software size. When a paper does not provide information on an attribute, the label "–" is used. The conclusions may be summarized as:

- *Study type*: Thomas et al. (1997) and Selby (2005) do not discuss the study type. For the others, we have shown the study type both from the authors' and the review's perspective. The main differences in classification are: (1) Quasi-experiments and experiments from the authors' perspective (Frakes and Succi 2001; Succi et al. 2001; Zhang and Jarzabek 2005) are classified as (exploratory) case studies and experience reports in this review because of the lack of clear hypotheses and the little degree of control applied by the investigators. (2) Two case studies from the authors' perspective (Lim 1994; Tomer et al. 2004) are classified as experience report and example application in this review. The term "case study" is often used in literature to cover all studies where some data on "cases" are presented. The review has identified four case studies (Thomas et al. 1997; Mohagheghi et al. 2004; Baldassarre et al. 2005; Selby 2005), three exploratory case studies (Frakes and Succi 2001; Succi et al. 2001[2]; Morisio et al. 2002), three experience reports (Lim 1994; Morad and Kuflik 2005; Zhang and Jarzabek 2005), and one example application (Tomer et al. 2004).
- *Publication channel*: Four papers are published in various conference proceedings and seven in journals, where *IEEE Trans. Soft. Eng.* has published four of the papers.
- *Year*: 2005 has been the most productive year with four papers.

Succi et al. (2001) and Tomer et al. (2004) have not reported programming language, and four of the papers (Tomer et al. 2004; Baldassarre et al. 2005; Morad and Kuflik 2005; Zhang and Jarzabek 2005) have not reported the size of products or the reusable assets, where in Zhang and Jarzabek (2005) it is not clear whether 4.5 KLOC is the total size or the mean size of applications. There is variation in domain and programming languages. Based on the given software size and our conclusions, we classified the studies according to their scale into:

- Small-scale studies (*S*): Five studies (Frakes and Succi 2001; Morisio et al. 2002; Tomer et al. 2004; Morad and Kuflik 2005; Zhang and Jarzabek 2005) cover a few reused software assets or small products.
- Medium-scale studies (*M*): Three studies (Lim 1994; Succi et al. 2001; Baldassarre et al. 2005) cover larger products than the first group but less or around 100 KLOC, and still the objects of study are few.

[2] Succi et al. (2001) can be classified as a case study as well, but research questions and hypotheses are not well-stated in the paper.

– Large-scale studies (*L*): Three studies (Thomas et al. 1997; Mohagheghi et al. 2004; Selby 2005) cover products with software size more than 100 KLOC or cover a large number of objects.

4.2 Reuse Approaches

Appendix B presents data on the reuse approaches. The definitions of terms and an overview for each field are given in Table 2.

Morisio et al. (2000) have a similar list for comparing projects with a few additional factors such as whether there exists an explicit reuse process or when the reusable assets are developed (on demand or beforehand). Most of the papers in this review did not include data on these factors.

4.3 Summary of the Section and Answering *RQ1*

All the 11 studies in this review are classified as observational studies with four case studies, three exploratory case studies, three experience reports and one example application. We conclude that our search has not returned any experiment or quasi-experiments in industry. Four studies are sister-project studies; comparing projects or products and in one case replicated product design. The closest to experimentation is comparing similar projects in size and domain, developed within the same company where developers have comparable skills or may be randomly assigned (Succi et al. 2001; Baldassarre et al. 2005; Morisio et al. 2002), or redeveloping a product with systematic reuse (Zhang and Jarzabek 2005). This review of literature has not found any study with random assignment of treatments to objects, and only Baldassarre et al. (2005) report random assignment of developers to the two projects. Zannier et al. (2006) report also that they did not find any example of simple or stratified random sampling in a sample of ICSE papers. The selection of objects (products or components) is due to access to data, which may be classified as investigator-selected or convenience sampling. On the other hand, Zannier et al. found the absolute majority of studies being self-confirmatory; i.e., the authors played a role in the development of the product of study. We did not find support for that in the review and only in three cases (Lim 1994; Mohagheghi et al. 2004; Morad and Kuflik 2005) the authors were employees of the companies. However, for several studies the relation of investigator to the case was not clearly stated.

The scale of the studies varied (small-scale studies are most represented) and so are the approaches to reuse, domain and reuse rate. Most studies involved systematic reuse or compared it with ad-hoc reuse. Units of reuse varied as well but only reuse of source code was measured. Only in the three small-scale studies, reused components were developed externally or before the application, and only Morad and Kuflik (2005) give an example of reuse of three OSS assets, where savings in person-hours are estimated. A few studies do not report size of the products or components, programming languages or characteristics of their reuse approaches.

Only two studies report data from several releases of a software product or projects over time; i.e., Mohagheghi et al. (2004) and Lim (1994), where the first study evaluates components in several releases of one software product and the second study reports productivity gains over several years. Selby (2005) has collected data for several years of development, but does not present the data as releases of the same products, only from the same environment. Most studies in this review are therefore cross-sectional, and long-term effects of reuse are understudied.

Table 2 Summary of reuse approaches

Approach	Definition	Findings
Development scope	Whether the reusable assets are from a source *internal* or *external* to the project (Frakes and Terry 1996). Examples of externally developed assets are those developed in other projects, COTS or OSS components.	In small-scale studies, reused assets are internal or external, while in the medium to large-scale studies, reused assets are all internal. Large-scale studies are characterized by internal reuse, architecture reuse and a systematic approach to reuse.
Technical approach	Refers to technical methods for implementing reuse. We started with the classification in Frakes and Terry (1996) but had to add several classes since their classification did not cover all the papers.[a] The classes used here are: Compositional; reuse of functions or subroutines (fine-grained); Reuse of templates which can be of any kind; *Reuse of software modules or components*; *Object-Oriented (OO) frameworks*; *Domain engineering* for product families; *Component-based reuse*[b] with adherence to component models such as CORBA/CCM/EJB; G*enerative programming*; *Reuse repository or library*, which can be generic or domain-specific, and can be combined with other approaches.	Compositional: Frakes and Succi (2001) and Succi et al. (2001); Reuse of templates: Baldassarre et al. (2005); Reuse of software modules or components: Thomas et al. (1997), Tomer et al. (2004), Morad and Kuflik (2005) and Selby (2005); OO frameworks: Morisio et al. (2002); Domain engineering: Mohagheghi et al. (2004) with a layered software architecture, and Zhang and Jarzabek (2005) with meta-components; Component-based reuse: Mohagheghi et al. (2004); Generative programming: none; Reuse repository or library: Thomas et al. (1997), Succi et al. (2001), Tomer et al. (2004), Morad and Kuflik (2005), and Baldassarre et al. (2005); Lim (1994) does not give any information on the approach to reuse.
Domain scope	Covers *horizontal reuse* of generic assets across domains such as general libraries, *vertical reuse* within a domain, and *architecture reuse* in domain engineering.	We found examples of all the three in the papers.
Reuse management	Refers to the degree to which reuse is done systematically, and may be *systematic* or *ad-hoc*. We consider having an explicit reuse program as a type of systematic reuse. Tomer et al. (2004) and Morad and Kuflik (2005) define a compromise between these two approaches as well, called *controlled reuse*, where candidate assets that may be reusable in the future are kept in a repository without putting effort in making them reusable. A future project that uses such asset will make a reusable asset out of it.	Nearly all papers cover systematic reuse or compare systematic reuse with ad-hoc reuse. Only Frakes and Succi (2001) is characterized with merely ad-hoc reuse.
Reuse initiation	Product families or in general any reuse program may be initiated in multiple ways (Krueger 2002): *Proactive* when companies can predict their reuse requirements and have resources to design all reusable assets up-front;	From the studies who have reported the approach to reuse initiation, Lim (1994) reported an incremental reactive approach, while Mohagheghi et al. (2004) and Zhang and Jarzabek (2005) reported an extractive approach.

Table 2 (Continued)

Approach	Definition	Findings
	Reactive when one or several product variations are developed at a time; *Extractive* when reusable assets are extracted from one or several products to make the product family baseline.	
Modification	Assets may be reused: *Verbatim* which means reusing an asset "as-is" In a black-box style; or *modified* in a white-box style to make an asset reusable for a new target.	Pure verbatim reuse: Frakes and Succi (2001), Mohagheghi et al. (2004), Tomer et al. (2004) for five assets, Morad and Kuflik (2005) and in the study of Succi et al. (2001) we concluded that reuse is most probably verbatim. Six studies have reported verbatim and modified reused, including Tomer et al. (2004) for two assets. Lim (1994) does not discuss this aspect.
Reuse rate	The size of reused assets divided by the software size.	It shows great variation in papers. While some companies consider 30% reuse as the acceptable goal of their reuse program Selby 2005), others consider this as low. We conclude that there is no golden figure and companies have different goals.
Unit of reuse	Varies from fine-grained functions to large-grained frameworks that are reused as-is or by architectural layers.	Although reuse of templates or meta-components is also reported, the quantitative data are on the source code level. Reuse of other assets than source code seems hard to be quantified or is covered by measuring source code.
Complementary factors	Refer to facilitators of reuse such as reuse training and management commitment.	Most papers have discussed some complementary factors.

[a] Their classification includes generative, compositional, in-the-small (component-based), in-the-large, indirect (through an intermediate entity) versus direct, carried-over (from one release to next) and leveraged (with modifications)

[b] The term "component-based reuse" is often used in literature to cover several approaches to reuse when systems are assembled of reusable "components" as independent units of production and acquisition. The reusable components may be retrieved from a repository, shared between products in a product family, or be obtained as OSS or COTS components. However, component-based reuse here refers to reuse of components that adhere to a particular component model, and target a particular component platform (Szyperski 2002), as a difference with other COTS or OSS software with no constraints on conformance to an architecture.

5 Answering *RQ2*—Which Metrics are Used for Reuse and its Effects?

This section presents the metrics used in the studies; except for the metrics related to cost-benefit analysis, which are few and are presented in Section 7 together with the findings.

5.1 Independent Variables—Reuse Metrics

While attributes of reuse approach presented in Section 4.2 such as *Development scope* or characteristics such as domain may be used as independent variables, most of them happen

to be fixed in the studies. The independent variables related to reuse are identified to be (see Table 3):

- *Development mode* is a two-level factor and refers to whether development happens with or without systematic reuse in a project. It is used in sister-project studies.
- *Component origin* is a multi-level factor and refers to whether a specific component (or any asset) is reused verbatim, slightly or extensively modified, or is newly developed. It is used in component-comparison studies.
- *Reuse rate* quantifies the amount of reuse in a project or sometimes within a component. Reuse rate may be used as a dependent variables as well, but not in the reviewed literature.

5.2 Dependent Variables—Reuse Effects

We analyzed all the dependent variables used in the papers and identified four major groups: metrics related to software problems (Table 4), effort and productivity (Table 5), software change (Table 6) and module level metrics (Table 7).

We use the term "problem" in the review covering errors, defects and faults when the distinction is not clear or when we refer to all of them. There is inconsistency and vagueness

Table 3 Independent variables and their definitions in the papers

Independent variable	Definitions
Development mode	Succi et al. (2001): Development with only a general library, or with a general and domain library.
	Morisio et al. (2002): Development with an OO framework or without it.
	Baldassarre et al. (2005): Development with ROD (reuse-oriented development process) or without it.
	Zhang and Jarzabek (2005): Games derived from a Product Line Architecture (PLA) or the same games without a PLA.
Component origin	Lim (1994): New and reused code versus new code only.
	Thomas et al. (1997), Selby (2005): Modules or components reused verbatim, with slight (<25%) or major (≥25%) modification, or newly developed.
	Tomer et al. (2004), Morad and Kuflik (2005): Components developed from scratch compared to when developed with ad-hoc, controlled or systematic reuse.
	Mohagheghi et al. (2004): Reused versus non-reused components.
Reuse rate	Thomas et al. (1997): Reused code relative to total code measured in the number of Ada statements.
	Lim (1994), Mohagheghi et al. (2004), Baldassarre et al. (2005): Reused LOC relative to total LOC.
	Frakes and Succi (2001), Succi et al. (2001): Compositional approach to reuse. External Reuse level (ERL) is the ratio of external lower level items (functions) reused inside a higher level item (file) over the total number of lower level items used. External Reuse Frequency (ERF) is the number of references to external lower level items (functions) reused inside a higher level item (file) over the total number of references. External Reuse Density (ERD) is the number of external lower level items (functions) called to LOC of a file.
	Morisio et al. (2002): Reuse level is the ratio between the size of what is reused from the framework and the total size in an individual application. Size is measured in Object-Oriented Function Points (OOFP).
	Selby (2005): Percentage of modules reused verbatim or with modification.

in the use of the above terms. Mohagheghi et al. (2006) identified three questions to answer: *what* is covered (problem appearance or its cause), *where* problems are (software vs. system, executable vs. non-executable software such as requirements or documentation) and *when* problems are detected (the detection phase). These differences are visible in Table 4:

- "Errors" are often counted for appearances of a problem. It may lead to changes in several modules or causes called for "defects" or "faults."
- Problems may be reported only for source code or all types of artefacts (executable and non-executable such as documents).
- Problems may be recorded pre-release, post-release or in both phases.

All of the papers in Table 4 have used metrics related to problems as *quality indicators*. Since the discussion of the relation between dependent variables and quality (quality-in-use or other views such as process quality) is often missing in the papers, we do not get into this discussion and refer to this as a threat to construct validity in Section 7.5. Lim (1994), Thomas et al. (1997), Frakes and Succi (2001), Succi et al. (2001) and Mohagheghi et al. (2004) have used counts of problems or its density, while Thomas et al. (1997), Morisio et al. (2002) and Selby (2005) have included rework effort or isolation and correction difficulty as quality indicators.

Table 5 shows metrics related to effort and productivity. Increasing productivity and decreasing development time or effort are often given as the main motivations for reuse. *Apparent productivity* is calculated by dividing the total size of software to the total effort spent, while *actual productivity* is calculated by dividing the size of newly developed code to the total effort. One inherent problem with this approach is that integration of reusable assets or their modification takes effort which is included in the total effort, while their size is not included. With reuse, apparent productivity increases obviously. We discuss an alternative approach to measure actual productivity in Section 8.2.

Reducing the number of changes or the size of modified code should improve maintainability of a product, which motivates the use of software change metrics in Table 6.

Module-level metrics are used in two ways as shown in Table 7: Sister-project studies have evaluated whether development with reuse reduces product complexity, while two component-comparison studies (Thomas et al. 1997; Selby 2005) have used these metrics to characterize reuse at module level.

5.3 Summary of the Section and Answering *RQ2*

We identified three independent metrics which are *Development mode*, *Component origin* and *Reuse rate*, while other attributes of reuse presented in Section 4.2 may also be used as independent variables.

Metrics used to measure reuse effects are divided in four groups: metrics related to software problems (used in seven papers), effort or productivity (eight papers), software changes (four papers) and software module characteristics (five papers). In addition to these, Zhang and Jarzabek (2005) have measured improvement in performance in terms of memory usage and speed in running time. The papers have used 22 different dependent metrics, with very few examples of a common definition. In many cases, metrics are not well defined either, especially for metrics related to software problems. The diversity of metrics and definitions makes comparison of quantitative results difficult. The tables in this section may help future studies in choosing metrics so that several studies use common or comparable metrics. This is one precondition for combining evidence systematically or performing any meta-analysis in the field.

Table 4 Metrics related to software problems

Metric	Papers and their definitions of metric
Defect density: the number of defects divided by the software size	Lim (1994): No definition of what a defect is. Size is in LOC. Succi et al. (2001): Customer Complaint Density (CCD) is the ratio of customer complaints to LOC and is actually post-release defect density.
Error density: the number of errors divided by the software size	Thomas et al. (1997): The source of error may be requirements, functional specification, design, code, or previous change. Errors may be reported in unit testing, system or acceptance testing. We assume these are pre-release problems. Size is in LOC. Frakes and Succi (2001): Errors are for source code but the definition and the detection phase are not given. Size is in LOC.
Fault density: the number of faults divided by the software size	Thomas et al. (1997): An error correction may affect more than one module. Each module affected by an error is counted as having a fault. Size is in LOC. Mohagheghi et al. (2004): Faults are causes of failure, studied for source code, and are detected in system testing and later phases; i.e. both pre- and post-release faults. Size is in LOC. Selby (2005): An error correction may affect more than one module. Each module affected by an error is counted as having a fault. Size is in LOC. Type of faults (what) and detection phase (when) are not given.
Rework effort spent in isolating and correcting problems	Frakes and Succi (2001): Subjective quality rating by developers based on the difficulty in debugging and maintaining software, from 1 to 10 (best). Thomas et al. (1997): Relative rework effort in person–hours divided by the number of statements spent in isolating and correction errors. Morisio et al. (2002): Quality of programming is defined as the relative development effort and rework effort to correct failures detected in acceptance testing; between 0 and 1 where 1 means no rework and 0 means rework effort equals development effort. Selby (2005): Fault correction error and fault isolation effort measured separately in person–hours per module.
Difficulty in error isolation or correction	Thomas et al. (1997): Isolation and correction difficulty for errors, where more than one day spent on isolation or correction indicates difficultly.
Source of error	Thomas et al. (1997): Source of error refers to where error was introduced such as requirement, design or code.
Error slippage from unit test	Thomas et al. (1997): Percentage of errors that escape unit test and are detected later.
Error type	Thomas et al. (1997): Error type may be procedural, interface or data.
Fault severity	Mohagheghi et al. (2004): Severity of faults for reused and non-reused components.
No. of faults	Selby (2005): The number of faults.

Table 5 Effort and productivity metrics

Metric	Papers and their definitions of metric
Development effort per module, asset, or product in person hours, days or months	Frakes and Succi (2001): Development effort in person–days spent per module. Tomer et al. (2004), Morad and Kuflik (2005): Development effort spent in different scenarios (new development, ad-hoc/controlled or systematic reuse) per asset in person–hours. Selby (2005): Average module development effort in person–hours, covering also corrections and changes. Zhang and Jarzabek (2005): Development effort of an application based on the reusable product line architecture and without it in person–days.
Apparent productivity	Lim (1994), Frakes and Succi (2001), Baldassarre et al. (2005), Selby (2005): LOC per engineering month or day or hour (including reuse). Morisio et al. (2002): Gross productivity is the size of application in OOFP (Object-Oriented Function Points) divided by development effort.
Actual productivity	Morisio et al. (2002): Net productivity is defined as the size of new code developed around the framework in OOFP divided by development effort. Baldassarre et al. (2005): New LOC divided by development effort in person–hours.
Time to market	Lim (1994): Reduction in time to market.
Design effort	Selby (2005): Percentage of development effort spent in design per module.

6 Answering *RQ3*—How are Quantitative Data Reported and Analyzed?

Appendix C shows how data are reported and analyzed in the eleven papers. We summarize the observations here.

Small-scale studies have used all the available data in the analysis and have mostly included the dataset in the papers. Except for Morisio et al. (2002) with hypotheses and a regression model, the other four small-scale studies have not defined hypotheses or applied inferential statistics. Medium and large-scale studies do not present all the data. However, data are fully analyzed and no sampling is done in these studies. In the three large-scale

Table 6 Metrics related to software change

Metric	Papers and their definitions of metrics
No. of changes	Frakes and Succi (2001), Selby (2005): The number of changes (enhancement or repair) to a module.
Change density	Mohagheghi et al. (2004): No. of requirement changes per LOC. Selby (2005): No of changes (enhancement or repair) per LOC.
Change in the amount of developed code	Zhang and Jarzabek (2005): Total size of product
Rate of modified code between releases	Mohagheghi et al. (2004): Size of modified code divided by the total size of code between releases (for components and the whole product).
Change implementation effort	Selby (2005): Effort per change in person–hours.

Table 7 Module-level metrics

Metric	Papers and their definitions of metric
Complexity of products	Morisio et al. (2002): The number of methods per unit net size (in OOFP).
	Baldassarre et al. (2005): Mean Cyclomatic Complexity (MCC) per artefact and for the system as a whole.
	Zhang and Jarzabek (2005): OO metrics for complexity.
Module characteristics	Thomas et al. (1997): Size in the number of Ada statements, the number of parameters as a measure of generality, the number of "with" as an indication of external dependency.
	Selby (2005): Size in LOC, assignment statements per LOC, the number of module calls excluding calls to utility functions per LOC, utility calls per LOC, input-output parameters per LOC, read and write statements per LOC, comments per LOC, MCC per LOC.

studies, the researchers have mined industrial databases and in two of them, data were inserted in relational databases for analysis. Appendix C also shows that the range of statistical tests is limited and there are no examples of data transformation (for example logarithmic transformations) in the studies.

Six papers have applied statistical tests and the authors have discussed preconditions such as normal distribution of data. However, when it comes to defining hypotheses and applying inferential statistics, we observe variances of the *null ritual* in four papers. Gigerenzer (2004) defines the null ritual in three steps:

1. Set up a statistical hypothesis of no difference or no correlation. Do not specify any alternative hypothesis.
2. Use 0.05 (or some other fixed value) as a convention for rejecting the null. Report the results as $p<0.05$, $p<0.01$ or $p<0.001$; whichever comes next to the obtained value.
3. Always perform this procedure.

The null ritual is a modification of the Fisher's null hypothesis testing, which may be summarized in the following three steps:

1. Set up a statistical null hypothesis. The null need not to be a nil hypothesis of no difference.
2. Report the exact level of significance and do not talk about accepting or rejecting hypotheses.
3. Use this procedure only if you know very little about the problem at hand. This procedure does not allow combining previous knowledge in inference; e.g., in contrast to the Bayesian approach.

Gigerenzer (2004) writes that statistical rituals eliminate statistical thinking and inferential statistics should be performed with care. Alternatives are Exploratory Data Analysis (EDA) techniques or reporting descriptive statistics and making conclusions without performing hypothesis testing. Even with well-defined null and alternative hypotheses, the selection of a 0.10 or 0.05 level of significance is a matter of personal choice, depending on whether a researcher is averse to missing a significant effect or to reporting a spurious effect. There is often no discussion of why a certain level of significance is selected in the papers or even why it varies within a single study.

Some papers in the review report the *p*-values while others do not or only report values over a certain threshold. Lim (1994) have discussed the results as significant for the company without applying inferential statistics, Succi et al. (2001) have reported *p*-values and considered the results as significant, and Mohagheghi et al. (2004) have

discussed practical significance for the company in terms of saved effort. Other papers have used fixed thresholds for discussing significance without reflecting on the practical significance.

7 Answering *RQ4*—What are the Findings and What Theory may be Developed Based on the Findings?

This section summarizes the findings in terms of reuse economics, quality and productivity benefits, qualitative findings and validity concerns.

7.1 Reuse Economics and Savings

An overview of metrics and findings related to cost-benefit models is given in Table 8. The cost of reuse is assessed in the costs of developing reusable assets and integrating them. No costs are evaluated for training, infrastructure for reuse or setting up reuse repositories. Savings are assessed in development and rework effort.

7.2 Findings Related to Quality and Productivity

The detailed findings related to quality and productivity in the four sister-project studies are shown in Appendix D. Kitchenham (2004) lists a set of criteria for quality assessment of studies. None of the studies claim to have selected their cases randomly from a population or have presented them as representative for a population. However, sister projects are claimed to be comparable with respect to domain, size, duration and developer skills (one developer in Morisio et al. 2002).

Five studies have compared reused components (verbatim or modified reused) with new code, sometimes within the same product and sometimes within a collection of products. We called these component-comparison studies. The studies of Tomer et al. (2004) and Morad and Kuflik (2005) are not included here since they only include data on effort savings. All data are analyzed in the studies. Appendix E summarizes the quantitative results of these studies.

In the sister-project studies, some control is applied by the investigators in the design of studies which ranks them higher in the chain of evidence. Component-comparison studies analyze available data on components with no control over the study. On the other hand, sister-project studies are all of small or medium scale, while three component-comparison studies have mined large industrial data bases. One observation of this review is that we have not found results in favour of no reuse or no systematic reuse, unless related to error correction difficulty (Thomas et al. 1997) and fault severity (Mohagheghi et al. 2004).

Two large-scale case studies; i.e. Thomas et al. (1997) and Selby (2005); have compared characteristics of reused modules with the non-reused ones. Both studies reported that modules reused verbatim were significantly smaller in size. Selby (2005) found that modules reused verbatim tended to be small, well-documented modules with little input–output processing. It also seems that these modules tended to be *terminal nodes*, because they had less interaction with other system modules but more interaction with utility functions. Thomas et al. (1997) reported that components reused verbatim from a domain library were smaller in size and had less external dependencies. There is however one difference: modules reused verbatim in Selby (2005) had simpler interfaces than other modules in terms of input–output parameters per LOC, while components reused verbatim

Table 8 Metrics and findings on reuse economics

Metric	Papers and findings
Cost of developing reusable assets compared to the non-reusable ones	Morad and Kuflik (2005): An average of 160–250% of developing non-reusable assets. Lim (1994): In HP, 111% of the cost of creating a non-reusable version. The cost is also evaluated for different phases of development. The most significant increases were in the investigation and external design phases to understand the multiple contexts in which the work product will be reused.
Cost of integrating reusable assets	Morad and Kuflik (2005): Integration of a black-box reusable binary component costs 1–3% of new development. Lim (1994): 19% of the cost of creating a non-reusable version.
ROI (savings/cost)	Lim (1994): ROI in product 1 was 410% over 10 years, and for product 2 it was 216% over 8 years. The model includes the time value of money, gross cost and savings of reuse (called Net Present Value). Reuse gave 42% reduction in time to market for product 2. Tomer et al. (2004): For all the seven assets, systematic or controlled reuse gave savings relative to new development. For five assets, systematic reuse gave savings between 42 and 81% compared to new development (measured in person–hours). For one asset, controlled reuse was 32% better than systematic reuse, while for others, systematic reuse would be best. Morad and Kuflik (2005): Savings of systematic reuse over new development is approximately 50%, and less for other reuse scenarios. For the OSS products, systematic reuse with adaptation would be best, although it is difficult to read exact savings from diagrams.
Rework savings	Thomas et al. (1997): Reuse via slight modification shows a 35% reduction in relative rework effort over newly created components, while verbatim reuse provides an 88% reduction. For these modes of reuse, the benefit of fewer errors clearly outweighs the cost of more difficult error correction for reused components. Mohagheghi et al. (2004): The lower fault density of reused components is estimated to reduce the total effort by 20%.
Break even point (recover creation costs)	Lim (1994): The break even point in one product occurred in the second year and in the other product in the sixth year. The number of reuses ranges from one to eight for different assets.

in Thomas et al. (1997) had more parameters than either modified or new components. Thomas et al. (1997) explain the difference to be related to Ada or FORTRAN approaches to reuse.

7.3 Combining the Results for Quality and Productivity

We have a range of quantitative results that we want to appraise and combine. Pickard et al. (1998) describe three methods for combining the results of empirical studies:

– *Combining the p-values of studies* which can reject a null hypothesis or fail to reject it, without giving any information on the actual effect.

- *Meta-analysis* when the studies have used comparable metrics and reported a quantitative measure of effect size.
- *Vote-counting* that does not depend on the actual effect size values and comparable metrics. Different outcomes of the hypotheses tests are categorized into significant positive effect, significant negative effect or non-significant effect. Each study then casts a "vote" in support of the above relationships and the numbers of votes are counted, thus becoming new scale that behaves like *p*-values. If the ratio of votes to the total number of studies is over a predetermined cut-off value, a relationship for the specific variable is identified. The method assumes that there is one underlying common phenomenon, for example when a single correlation coefficient is applied.

Each of the above methods has its requirements. The first one depends on *p*-values which are not reported in several studies. Meta-analysis requires homogenous studies and comparable metrics, while the studies in this review vary in type and metrics. Vote-counting requires an underlying common phenomenon but it allows testing very weak hypotheses. However, it may be the only method applicable when there are different metrics for a phenomenon or the reported information is very limited.

We decided therefore to perform a modified approach of vote-counting by categorizing the findings in "significant positive," "significant negative," "positive," "negative," and "no relation." This way, we can evaluate *the weight of evidence*. By the weight of evidence, we mean the extent to which empirical results are consistent across a variety of studies (Pickard et al. 1998). We add the scale of the studies to evaluate whether reuse scales up, and finally significance in our vote-counting covers both practical and statistical significance depending on which one is discussed in the papers. Vote-counting is also discussed in Mohagheghi and Conradi (2006).

Table 9 shows a summary of findings. The dependent metrics are ordered after their popularity as given in the column "Metric included" (from 11 studies). In Table 8, "+ +" means a significant positive effect of reuse, "+" means positive effect, "**0**" means no relation or inconsistent results, "–" means negative effect and "– –" means significant negative effect of reuse. Note that Frakes and Succi (2001) do not discuss significance due to small sample size, Lim (1994) discusses practical significance, three studies (Tomer et al. 2004; Morad and Kuflik 2005; Zhang and Jarzabek 2005) are experience reports or example applications without discussion of significance, while the remainder of studies discuss statistical significance (and in cases practical significance as well). The three last columns in Table 9 show summary statistics indicating the number of studies that include a metric and how often the results were significant positive or negative.

Three dependent metrics are not included in Table 9 where the results were difficult to interpret (sources of error, decrease in time-to-market due to reduced development effort, and design effort). More than half of the dependent metrics are only used in single studies. The relations in Table 9 can be summarized for the independent or dependent metrics. When summarized for the independent metrics:

- *Development mode*: When comparing development with systematic reuse to development without it across projects in four studies (Succi et al. 2001; Morisio et al. 2002; Baldassarre et al. 2005; Zhang and Jarzabek 2005), significant increase in apparent productivity is reported in two of them. In case of actual productivity and complexity of products, the results are inconsistent. Other benefits are only reported from single studies. From the above studies, Zhang and Jarzabek (2005) is an experience report with no discussion of significance.

Table 9 Vote-counting for reuse effects on software quality and productivity

	Frakes and Succi (2001)–S	Morisio et al. (2002)–S	Tomer et al. (2004)–S	Morad and Kuflik (2005)–S	Zhang and Jarzabek (2005)–S	Lim (1994)–M	Succi et al. (2001)–M	Baldassarre et al. (2005)–M	Thomas et al. (1997)–L	Mohagheghi et al. (2004)–L	Selby (2005)–L	Metric included	Significant positive	Significant negative
Defect, error or fault density, pre/post release or undefined (*u*)	+ *u*					++ *u*	++ post		++ pre	++ both	++*u*	6	5	0
Development effort per module, asset or product in person–hours/days/months	+		+	+	+						++[a]	5	1	0
Rework effort (effort spent on corrections) per LOC or per module (M)	+	++							++ LOC		++ M	4	3	0
Apparent productivity	0[b]	++				++[c]		++[d]				4	3	0
Complexity of products		0			0			++				3	1	0
Actual Productivity		++						0				2	1	0
No. of changes to a software module	+										++	2	1	0
Change density										0[e]	++[f]	2	1	0
No. of faults											++	1	1	0
Difficulty in error correction									– –			1	0	1
Difficulty in error isolation									0			1	0	0

Table 9 (Continued)

	Frakes and Succi (2001)–S	Morisio et al. (2002)–S	Tomer et al. (2004)–S	Morad and Kuflik (2005)–S	Zhang and Jarzabek (2005)–S	Lim (1994)–M	Succi et al. (2001)–M	Baldassarre et al. (2005)–M	Thomas et al. (1997)–L	Mohagheghi et al. (2004)–L	Selby (2005)–L	Metric included	Significant positive	Significant negative
Fault severity										−−		1	0	1
Error slippage from unit test									0[g]			1	0	0
Amount of developed code					+							1	0	0
Rate of modified code between releases										++		1	1	0
Change implementation effort											++[f]	1	1	0
Performance (memory usage and speed)					+[h]							1	0	0

[a] Comparing of projects showed that average development effort per module decreased with more reuse, although not significantly

[b] The results across four small datasets were inconsistent, with some being positive and some not

[c] Lim reports that apparent productivity increase for two projects due to component reuse led to shorter time-to-market. Also increased reuse rate was positively correlated with increased apparent productivity

[d] Apparent productivity was less with reuse in the first period since the repository had to be populated. After that, the reuse-based process was more productive that conventional development

[e] The number of requirement changes per component does not show significant difference for reused and non-reused components

[f] We could not find the exact figures in the paper but the results were significant at 0.05 level

[g] Nearly two thirds of the errors escaped unit testing in all types of reuse (verbatim, slightly or extensively modified and new), except for slightly modified code with 43%

[h] The reusable product line architecture included optimization strategies that could propagate to the applications and improve their performance

- *Component origin*: In component-comparison studies, systematic reuse (either verbatim, with slight modification or mixed with new code) is related to significant decrease in problem density in four studies (Lim 1994; Thomas et al. 1997; Mohagheghi et al. 2004; Selby 2005). From these, Lim (1994) is an experience report where only practical significance is discussed. Systematic reuse is also related to significant decrease in rework effort in two studies (Thomas et al. 1997; Selby 2005). Decrease of development effort per module or asset is studied in Frakes and Succi (2001), Tomer et al. (2004), Morad and Kuflik (2005) and Selby (2005), but only Selby (2005) discusses significance. Note that the definitions of metrics vary across studies. Other significant positive impacts are verified by single studies or the results are inconsistent. Two significant negative effects are also reported by single studies: Thomas et al. (1997) report significant difficulty in error correction for verbatim reused code since probably easier errors are already removed from reused code, and because developers have a greater familiarity with newly created components. We add that sometimes, it may be difficult to find the cause of a problem when a component is reused since the problem may be in integration or interaction with other components. Mohagheghi et al. (2004) report significant higher fault severity of reused components because problems in these components would lead to service unavailability or restarts. However, severity or difficulty ranking are often subjective and may not reflect the impact or importance of modifications.
- *Reuse rate*: The studies of Frakes and Succi (2001) and Succi et al. (2001) are correlational; meaning that they evaluate the relation between reuse rate and the dependent variables. In the above studies, increased reuse rate is related to decrease in problem density, although not significant in Frakes and Succi (2001). Lim (1994) and Selby (2005) report some results related to reuse rate that are given under "Notes." The increase in apparent productivity with increasing reuse rate is reported as significant in Lim (1994), while the four datasets in Frakes and Succi (2001) were inconsistent on the relation between the External Reuse Level (ERL) and apparent productivity.

We can also summarize the findings horizontally for each dependent metric, where we only summarize rows with several reported results and also give explanations from papers on why reuse is related to the outcome:

- *Defect, error or fault density is significantly reduced* with introducing systematic reuse as confirmed by several medium and large-scale studies of different types, although the definitions of metrics and whether they study pre or post-release problems vary. Succi et al. (2001) observed that customer complaints reduced with increasing reuse rate when a domain library was in place, and not with reuse of a generic library, meaning that it is systematic reuse that had a positive impact. Reused components may be designed more thoroughly and be better tested, since faults in these components affect several products and the prevention costs are amortized over several products (Mohagheghi et al. 2004). Also because work products are used multiple times, the accumulated defect fixes results in a higher quality work product (Lim 1994). It is interesting to notice that reuse with extensive modification does not provide the reduction in problem density that the other modes of reuse. Selby (2005) reports that the modules reused with major revision had the highest fault correction effort, highest fault isolation effort and highest change correction effort, due to the loss of original design and abstractions. Thomas et al. (1997) did not observe any significant difference in defect density of extensively modified components versus new code, and modified components had more design errors.

- *Decrease in development effort per module or per asset* is verified in four small-scale studies without discussion of significance, and in one large-scale study with significant results; i.e., Selby (2005). Selby means that reuse leads to less effort spent in design because creation of a new module requires the creation and evaluation of a new design, while reuse may require a walkthrough of existing design.
- *Rework effort is significantly reduced* with systematic reuse. (1997) evaluated rework effort per LOC, Selby (2005) per module and Morisio et al. (2002) relative to the development effort. In the case of Selby (2005), rework effort is lowest for modules reused verbatim, but these modules are also smallest in size. The difference is also significant for modules with slight revision. Evidence is obtained both from sister-project and component-comparison studies of different scales. The fewer number of problems or lower problem density reduces the rework effort (Thomas et al. 1997). Morisio et al. (2002) write that more difficult tasks are probably already performed by framework designers.
- *Apparent productivity improves significantly* with systematic reuse (Lim 1994; Morisio et al. 2002; Baldassarre et al. 2005), and the positive relation with reuse rate is reported in Lim (1994). Evidence is obtained from two small or medium-scale sister-project studies and one medium-scale component-comparison study. Because the work products have already been created, tested and documented, apparent productivity will increase. However, increased productivity does not necessarily shorten time-to-market because reuse must be used effectively on the critical path of a development project (Lim 1994).
- *Results regarding actual productivity are inconsistent*. Baldassarre et al. (2005) report that actual productivity was not significantly different between the two projects developed with systematic or ad-hoc reuse. Morisio et al. (2002) report increase in actual productivity and relates it to learning.
- *Results regarding complexity are inconsistent*. Baldassarre et al. (2005) report significant decrease in complexity with reuse and mean that without systematic reuse, a software system becomes more complex and more difficult to maintain. Zhang and Jarzabek (2005) and Morisio et al. (2002) did not observe any decrease in complexity when applying systematic reuse. Note that the definitions of the metric varied as shown in Table 7. All the studies are sister-project studies.

7.4 Qualitative Findings

Although this review focuses on studies with quantitative findings, we give an overview of a few reported qualitative findings in the papers here:

- Reuse allows a company to use personnel more effectively because it leverages expertise (Lim 1994). More experienced personnel can be assigned to develop the reusable assets.
- Selby (2005) reported that larger projects reuse more with modification than smaller ones since scale may motivate reuse.
- Mohagheghi et al. (2004) reported that a reusable architecture leads to clearer abstraction of components. Reuse and standardization of software architecture and processes allowed also easier transfer of development in the conditions of organizational changes (Mohagheghi and Conradi 2007).
- Morad and Kuflik (2005) reported that reuse adoption was slower than expected and the management hesitated to assign resources to the reuse team.

7.5 Validity Threats Discussed in the Papers

The validity of a study is the degree of confidence in inferences made from the data; i.e. inferential quality. Only five papers have discussed validity threats at all: Thomas et al. (1997), Succi et al. (2001), Morisio et al. (2002), Baldassarre et al. (2005) and Mohagheghi et al. (2004). We discuss the four classes of validity threats below.

Construct validity is concerned with whether the selected metrics reflect the intervention and effects; i.e., "right metrics." Morisio et al. (2002) discuss that size and complexity were known from earlier studies while net rework effort was selected as a quality indicator since it is integrated in the effort model. Mohagheghi et al. (2004) discuss that fault density is used to compare the quality of components within the same environment, and is widely used. In the same study, the rate of modified code between releases (code volatility) was selected since less modified code has fewer faults. Succi et al. (2001) found a high correlation between two of the metrics (External Reuse Level and External Reuse Frequency), meaning that they are not orthogonal. None of studies have validated the selected metrics for their discriminative power, predictability or repeatability as recommended by Schneidewind (1992). Selection of metrics is often constrained by the available data. The relation between the selected metrics and quality is not well-discussed either. For example, Fenton et al. (2002) discuss that the number of detected problems is a function of both test effectiveness and potential problems, and few problems pre-release may indicate poor testing or high quality software. We notice that 3/6 studies using problem density have not even discussed whether they count pre- or post-release problems.

Conclusion validity for statistical analysis is concerned with whether the relationship between intervention and outcome is of statistical significance; i.e. "right analysis." As discussed in Section 6, the conclusion on significance is in many cases based on fixed thresholds.

Internal validity is concerned with whether the observed relation is a causal one or "right data" are collected, and is also a condition for external validity. It is difficult to discuss cause-effect without manipulation in controlled experiments and removal of confounding factors. Another view of causality given by Mill and discussed in Gregor (2002) is that: (a) the cause has to precede the effect in time, (b) the cause and effect has to be related, and (c) other explanations of the cause-effect relation have to be eliminated. In cross-sectional studies in which all the data are gathered at one time, the researcher may not even know if the cause precedes the effect (Shadish et al. 2001). Adding comparison groups and pre-treatment observations to case studies clearly improves causal inference (same place). We identified three sister-project studies with some degree of control applied by the investigator (Succi et al. 2001; Morisio et al. 2002; Baldassarre et al. 2005). These studies discuss internal validity in relation to the study design, as shown under confounding factors in Appendix D. The question is whether this discussion is enough for establishing causality. Other confounding factors discussed in the papers are the impact of size (Thomas et al. 1997; Succi et al. 2001; Mohagheghi et al. 2004), complexity of modules or their interfaces (Thomas et al. 1997; Selby 2005), programming languages (Mohagheghi et al. 2004), developer skills (several studies), learning (Morisio et al. 2002) and differences in the functionality of components (Mohagheghi et al. 2004). The three experience reports (Lim 1994; Morad and Kuflik 2005; Zhang and Jarzabek 2005) and one example application (Tomer et al. 2004) do not include discussion of confounding factors.

External validity of the results should be discussed to evaluate whether the results are generalizable to a population, other contexts ("right context") or to theory. Studies in this review are from industry, although cases are not claimed to be representative. Succi et al. (2001) write that the major results described in the paper can be extended to the underlying

normal population (what is it?). The major limitation to external validity in Morisio et al. (2002) is discussed to be the employment of a single subject in the study, who cannot be representative for all developers. Even with valid results, the set of projects with similar size, domain, language or development methods is not well defined or so small that generalization outside the companies is difficult, and similarity of projects is not easy to assess. Industrial studies in this review have not been performed by taking samples from a population or selecting cases based on pre-defined criteria, other than access to data. Other views of external validity should therefore be sought. Lee and Baskerville (2003) propose generalization to theories or models (an example in Mohagheghi et al. 2004). Another view of generalization is evaluating the weight of evidence in the context of reader's experience and how the results may be valuable in enhancing the evidential force to encourage a technology or approach.

In addition to the inferential quality, it is necessary to discuss data quality and reliability. Only Mohagheghi et al. (2004) discuss missing and inconsistent data due to the process of reporting problems and changes that do not enforce developers to enter necessary data. We did not find a discussion of data quality in the other papers. Pfleeger (2005) discusses some characteristics in evaluating credibility of single studies such as sensitivity to errors, quality and duration of observations, the expertise of those conducting and reporting studies, and their interest in the results. Yin (2003) recommends being careful to ascertain the conditions under which documents or archival records are generated and for which purpose.

7.6 Summary of the Section and Answering *RQ4*

We summarize the findings in several aspects:

Reuse savings In spite of the variety of cost-benefit models (Lim 1996 compares 17 of them), we have little empirical evidence from industry on the actual economic benefits of reuse. The studies of Tomer et al. (2004) and Morad and Kuflik (2005) have compared scenarios of reuse, Thomas et al. (1997) and Mohagheghi et al. (2004) have evaluated savings in rework effort, while Lim (1994) is an exception here with presenting long-term data on savings.

Organizational impacts Quantitative findings are rarely reported together with organizational impacts and feedbacks to industry. We may think of several reasons: Most analyses are performed by outsiders, data were collected or analyzed after the projects had finished, and the researchers did not perform long-term studies. The three studies that have analyzed large reuse programs (Thomas et al. 1997; Mohagheghi et al. 2004; Selby 2005) have actually mined industrial data repositories. Even in the sister-project studies, it is not reported whether the better performance of a reuse-oriented process had any impact on industry decisions or their development processes. Pfleeger (1996) writes that quality metrics per se, such as performance measures of defect rates, make no explicit strategic or economic statement. It is important to relate the results to the industry settings since "quality by itself is no longer a strategy that will ensure a competitive advantage. We must use quality intelligently, as one component of the overall business strategy."

Reusable assets When reusable assets are on the level of modules or functions, smaller and less dependent software modules are more often reused as confirmed by two papers. However, we found examples of reuse of large-grain building blocks as well, such as components in a layered architecture (Mohagheghi et al. 2004), product line architectures (Zhang and Jarzabek 2005) or OO-frameworks (Morisio et al. 2002). For large-grain

components, the researchers write that reusable assets may encapsulate more difficult design; i.e., leverage of expertise.

Combining the quantitative results We applied the vote-counting approach to combine the quantitative results, where the goal was to identify where positive or negative results are reported, and where they are significant. When it comes to productivity, significant increase in apparent productivity is verified by two sister-project studies of small or medium scale, and the study of Lim (1994). Results regarding actual productivity are few and inconsistent. Significant decrease in development effort per module, asset or product is reported in one study, while four small-scale studies have evaluated it and reported positive results, although not significant. Reuse led to significantly lower problem density and less rework effort, verified in several studies of all scales and in both sister-project and component-comparison studies. There are other benefits that are verified in single studies and a few disadvantages as well (but not as primary results of studies).

Ranking of evidence Kitchenham (2004) ranks evidence obtained from studies in five classes, where *1* is the highest and is assigned to the evidence obtained from at least one properly designed randomized controlled trial, and *5* is the lowest and is assigned to the evidence obtained from expert opinion based on theory or consensus. We did not find examples of experiments but when the degree of control is used to rank the evidence, three of the sister-project studies (Succi et al. 2001; Morisio et al. 2002; Baldassarre et al. 2005) would rank higher. Another step in appraising evidence is to evaluate how studies have handled validity threats. We found no analysis of metrics regarding their construct validity. Metric selection criteria was either not given or based on availability of data. Object selection was based on convenience or the criteria were not described, and in most of the studies, relation of the author to the case was not discussed. We do not criticise studies for selecting objects, subjects or metrics based on convenience, but for the absence of discussion regarding the selection process. Few papers discuss validity threats and confounding factors or seek alternative explanations. The quality of data is generally not discussed and we conclude that there is much room for improving design, analysis and reporting of studies.

Theory in software reuse The need for useful and sound theories has never been emphasized more, but there are few examples of what constitutes theory in software reuse research. Gregor (2002) extends the definition of theory from explaining "why" to cover different stages in research. The goals of theory as defined by Gregor and the contributions of the review are summarized in Table 10.

8 Answering *RQ5*—What are the Shortcomings in Reuse Research?

We discuss the question in three sections: evaluating ROI, measurement issues and other ideas for future research.

8.1 Evaluating ROI

In a recent paper by Frakes and Kang (2005) on the state of research on reuse and its future, the authors write that "much data on the effect of reuse on important variables such as cost

Table 10 Theory in software reuse based on the review results

Goal of theory	Definition	Examples of research methods	Contributions of the review
Analyzing and describing	When very little is known about the phenomenon under question. A contribution should provide logical descriptions and classifications.	Analysis of existing evidence; Empirical observation.	This review analyzes evidence on reuse and classifies studies in several aspects; thus building this type of theory where there have not been any systematic review before.
Understanding	Explains how and why something happened. Judgment regarding the contribution is made on the basis of whether new or interesting insights are provided, and on the plausibility, credibility and validity of the arguments made.	Case studies; Surveys; Ethnographic and interpretive field studies.	We should answer whether there is a causal relationship between reuse and the observed effects, or how reuse has contributed to the effects. We gave the explanations from the papers when discussing the evidence. However, several observations are not followed by discussions.
Predicting	Describes what will be, without necessarily understanding the causes. The existence of correlation between two variables has a prediction value but do not describe why something happened.	Analyzing past data.	We have found examples of regression and correlation, but no examples of prediction models. One aspect of prediction is what types of assets are most probably reused which was discussed in Section 7.2.
Explaining and predicting	That says what is, why and what will be.	All research methods including case studies and experiments.	We have not found well-built examples of this type of theory in the reviewed literature. We think that the reason is that explaining and predicting needs longitudinal studies, experimentation or large sample sizes.
Design and action	Says how to do something.	Action research	The review does not include this type of papers but have identified findings of this type of theory. Papers relate systematic reuse and not ad-hoc reuse to the benefits, and reuse in large is applied by architecture reuse (Mohagheghi et al. 2004; Selby 2005).

of software production, time to market and project completion time have also been reported, though these studies tend to be quasi-experimental." We have not found support for this claim in the reviewed literature. There may be several explanations:

- Researchers are not interested in cost-benefit analysis. We think that the extensive amount of literature and models on reuse economics rejects this explanation.
- Companies collect little data that may be used in credible cost-benefit analysis. It is possible to think that once the decision on reuse is taken based on some initial estimates of costs and benefits, companies do not collect data to evaluate such estimates. It may be difficult to account for investments in reuse such as infra-

structure, training or making assets reusable. Another reason may be reliance on expert opinion.
– Data are often analyzed by outsiders and not the company personnel. Outsiders have limited access to data on reuse investments, while industry either does not evaluate or does not report evaluations of economic success or failure of reuse programs.

Evaluating the above explanations, suggesting other ones or performing realistic ROI analysis on reuse are subjects for future research. *Sustainability* is related to making better links between reuse and corporate strategy (Frakes and Kang 2005).

8.2 Measurement Issues

Frakes and Terry (1996) have presented a survey of reuse metrics and models and classified these in six types: cost-benefit models, maturity assessment models, amount-of-reuse metrics, failure modes models to find reuse impediments, reusability assessment models, and reuse library metrics. This review only found examples of metrics related to cost-benefit models (Table 8), the amount-of-reuse metrics (reuse rate in Table 3), and reusability assessment models (module-level characteristics in Table 7). A comparison of metrics shows several challenges:

– *Measuring reuse of other assets than code and effort spent on reuse*: Software architecture, design, test cases and templates are reused but their reuse rate, effort to make them reusable or adapt them to a context are not quantified. One obvious reason is that changes in other assets than code is not measurable by tools and involves human judgement.
– *Using comparable metrics*: Few studies have used comparable metrics. Future studies should define their metrics precisely compared to the ones already used, and if possible use identical or comparable metrics.
– *Validating metrics*: Metrics should be evaluated by assessing their relation to quality (quality is defined in so many ways, but everybody agrees that it is made up of a collection of attributes where being fault free and delivered on-time are a few of them (Glass 1997)), prediction value or their power of discrimination. Such analysis needs data from several projects or over time. Using expert opinion is also an alternative when such history does not exist; see for example Li and Smidts (2003).
– *Measuring actual productivity*: Actual productivity is often calculated by dividing size of new code to total effort (see Table 5), but this does not show the productivity of a project. One solution to this problem is to define the size of developed software as the size of new code plus the *equivalent size of reused code*. Then actual productivity may then be measured by dividing the size of developed software by total effort. New code covers also glue-ware written to integrate components or add-ware to modify components or make them reusable. COCOMO II includes a model to estimate the equivalent size of reused software depending on factors such as design and code modification rate and the understandability of reused software (Boehm et al. 2004). Other models may be developed for the context.

8.3 Other Gaps for Future Research

Frakes and Kang (2005) propose future research to concentrate on techniques such as better presentation of reusable assets, education on reuse in universities and training in industry,

sustainability of reuse programs, identifying and validating metrics of reusability, and relationship of reuse and domain engineering to newer software development processes such as agile methods. The results presented in this review highlight the following gaps for future research (other than ROI and measurement issues discussed before):

- *Longitudinal studies* over releases to validate metrics and conclusions, identify costs or break-even points, and organizational impacts.
- *Reuse process with integrated metrics for reuse*. Mohagheghi et al. (2004) discuss the lack of reuse metrics in the development process. Baldassarre et al. (2005) presented a reuse-oriented process as part of a full reuse maintenance model. Other studies do not give any description of the relation between reuse and software development processes.
- *Improving the state of data collection and analysis in industry*. The fact that data collection in several studies needed development of additional tools and restoring of data shows the gap between academia and industry in collecting data. Even for problem reports that are collected by all companies, there are major concerns regarding the quality of data and its prediction value. In many cases data are given to researchers in formats that are not analyzable due to the limitations in commercial tools. This observation confirms that the collected industrial data are not analyzed by industry to a large extent (Mohagheghi et al. 2006).
- *Study reuse of COTS and OSS components*. The review mainly found examples of internal development for and with reuse. A recent large survey in Norway, Italy and Germany showed that 1/3 of ICT companies practiced some OTS-based development (Conradi et al. 2005), and a survey of 61 ICT companies in Norway (although a non-representative sample) showed that 68% of them are using COTS or OSS components (Sommerseth 2006). Recently, a report on defect density of over 30 widely used OSS products were published (Coverty 2006), but we don't have observations from industry.
- *Developing theories and models* in addition to presenting results. We believe that this review has taken a first step by analyzing evidence and collecting explanations. Future studies should start with theory or models of inputs and outputs, be more explanatory regarding their results by combining quantitative evidence with qualitative observations, and discuss validity threats. They should seek for multiple explanations and insight.

9 Lessons for Future Studies

The review of literature from different views brings a range of issues to the forefront for improving the state of research.

9.1 Defining Context and Data to Report

One important question is how much to report on the context to allow comparison of studies. The guidelines for empirical research by Kitchenham et al. (2002) may be used in design and reporting of studies. Based on the review results, we have summarized the minimum for reporting in Table 11.

9.2 Data Analysis

Industrial studies are mostly of the observational type. Researchers use data that are collected in the industrial settings and have usually little control over the environment or

the data collection procedures. Researchers often apply statistical inferential techniques on this collection of non-random data with questionable quality. There is more control in sister-project case studies but again the settings are not artificial to control all the variables. We discussed in Section 6 that variations of the Fishers's null hypothesis testing are the dominant method for inference. Some improvements to the analysis may be suggested.

Hypotheses statement Alternative hypotheses are only stated in few papers, while others assume the alternative hypothesis to be a state of difference between means without mentioning it. Papers should be more explicit on that. There are examples in Morisio et al. (2002) and Mohagheghi et al. (2004) where the expected outcome is stated as the hypotheses of the studies.

Testing hypotheses P-values are only reported in three studies which make taking independent conclusions difficult for readers or for combining the results in a meta-analysis. Gigerenzer (2004) and Wang (1993) recommend reporting *p*-values rather than the accept-reject method based on a fixed threshold level. We have not found a study where the experimental approach of defining sample size and type I and II errors beforehand was done, or the power was calculated. If applicable, see (Dybå et al. 2006). However, effect size may be discussed in some cases even without experimental design. Effect size is the

Table 11 Reporting context and data

Attribute	Description
Study objects or units of analysis, and subjects	Product and component size, development effort, size of reused assets, programming language, domain or type of application, internal goals for reuse or data on company baseline (if any), and skills of developers.
Criteria applied to select objects and subjects	Random, by convenience or other criteria.
Relation of authors/investigators to the case	This is important for evaluating objectivity.
Type of study	Information on the degree of control (intervention or observation), pre-test or post-test measurement, and whether there exists of a control group or the method of comparison if a study is comparative.
Reuse approach	Development scope, technical approach, reuse management, reuse initiation, modification, reuse rate, what is measured in reuse, any complementary factors including human factors proposed in Morisio et al. (2000), when reusable assets are developed, explicit reuse process and reuse team. See Table 2 for definitions.
Criteria for selection of metrics	Related work, precise definitions and how they should be linked to the propositions of the study. One improvement would be to select popular metrics that are already used in other studies.
Data collection process	Description of how data are collected, by whom, a discussion of the quality of data or any measurement problems or biases and how these are handled. Do data include subjective assessments, missing or invalid points or outliers that affect the results? Sources of data (see Section 2.3) such as archival records or participant-observation (the reporter participates in the case as a staff member) should also be reported.
Presented data	An overview of data in terms of the number of data points, means, medians, standard deviations or variances, or in box plots and diagrams, differences between mean values if comparison and effect size, the confidence intervals and the period of data collection.

difference of mean values divided by the pooled standard deviation and gives information on the actual observed difference.

Evaluating the results Researchers often take a conclusion of rejecting or not rejecting a hypothesis which is expected of them to do, but this should not merely depend on *p*-values. Descriptive statistics and discussion of practical significance in the settings are fundamental. One reason is that selection of a 0.05 or any other level is often subjective. A second reason is that practical significance is a balance between effects and costs. We found earlier that there is no golden figure for the reuse rate and the same is true for improvements in productivity or problem density. While a 5% reduction in problem density may be considered as significant in one setting, it may be considered as too low in another setting relative to the investments on reuse.

9.3 Explaining the Results

We observed that few studies have discussed why a result is observed or try to establish causality by eliminating alternative explanations. As discussed in Shadish et al. (2001), it is especially important in non-experimental designs to assess alternative plausible explanations. The results should also be discussed relative to internal goals (Lim discusses that in one case, reuse rate exceeded the internal goal after a few years), previous data, feedback from industry, the authors' experience, or impacts on a company's reuse process and decision-making.

9.4 Evaluating Contribution for External Validity

Based on the goals of theory depicted in Table 10, reporting from single case studies can have one of the following goals:

- Identifying commonalities and differences between settings for the purpose of analyzing and describing, and predicting the impacts for the outcome.
- Providing new or interesting insights for the purpose of understanding or explaining.
- Theory development or verification of theory either for explaining, predicting or action.

We may therefore ask the following questions to evaluate the contributions: Do the results strengthen or weaken our previous theories or expectations? Does a study have characteristics that make it unique for identifying new variables, gaining new insights or expecting atypical results? Does a study fill a gap or answer a question where we have little evidence, for example related to actual productivity or reuse of externally-developed components? Can we make conclusions about the theory or outcome and not about the population; for example, extensively modified components will probably be more defect-prone since the original design is significantly altered.

10 Conclusion

This review examined empirical studies in industry published between 1994 and 2005. The contributions of the review are identifying the extent and type of empirical research on reuse in industry, identifying context parameters, analyzing the metrics, combining the

findings, seeking for explanations and theory building, identifying gaps for future research, and suggestions for improving empirical studies in this field.

General observations Industrial studies may assure a high degree of relevance since the settings are not artificial and the developers are professional. On the other hand, several conditions are not controlled by the investigators. This fact does not explain the low quality of much research in the field. We found several papers that lacked information on their study design, research questions and hypotheses. Generally, metrics were poorly defined and there were few discussions on the quality of data. Also, the insufficient discussion of results in many studies and the lack of attention to establishing causality create serious doubts about the validity of conclusions. Little research has been done on important questions regarding sustainability of reuse programs and their impact on organizations and businesses. Since experimentation does not seem to be applicable in subjects such as software reuse that need context and observation over long time, we must strive to improve the quality of observational studies. Journals and conferences can play an important role here by requiring higher quality from the submitted papers.

Findings We performed the most basic form of combining empirical evidence which is vote-counting, showing also when the results are non-significant or when significance is not discussed. The best would be to have adequate number of studies to divide them depending on the study types. In spite of the concerns discussed above, the review found evidence for significant positive effects of reuse on:

- *Software quality*: There is positive and significant evidence on lower problem density (defect-, error- or fault density) and effort spent on corrections (rework effort) with introducing systematic reuse in industry. Problem density and rework effort may have been selected because industry collects problem reports, and relates product quality to the reported problems and effort spent on correcting them. A ranking of software engineering metrics by experts showed that problem density was among the top three measures in all phases of development (Li and Smidts 2003). The relation between the selected dependent metrics and quality needs better validation to improve construct validity.
- *Productivity*: There is positive and significant evidence on apparent productivity gains in small and medium-scale studies. Increasing productivity has been one of the main motivations for reuse. The results for actual productivity are inconsistent and the definition of metric is also problematic. Further studies are therefore necessary to evaluate productivity gains.

The scale of the studies and other characteristics such as application domain and the approach to reuse varied. The variation shows that reuse works in various situations and is practiced in multiple ways. Software industry has few standardized metrics and comparing studies may lead to a progress in this area.

One important question is to identify contexts where reuse is beneficial and how reuse should be applied to observe the benefits. Evidence collected from the studies suggests that:

- It is verbatim reuse and reuse with slight modification that results in significant lower problem density, development or correction effort.
- When reusable assets are on the level of modules or functions, smaller and less complex ones may be reused more often. For large-scale reuse, the reusable assets may incorporate difficult design decisions.

– Medium and large-scale reuse programs invest on developing the reusable assets internally.

Gaps The intention of the review is to assist decision-makers about reuse investments and future research to focus on unsolved issues. It highlights gaps in empirical research to be:

– *For researchers*, verifying economic returns of reuse, using comparable and consistent metrics for measuring reuse and its effects so that empirical evidence can be collected and appraised in a more effective way, improving analysis and statistical thinking, and improving the state of research design and reporting are major challenges.
– *For industry*, improving tools and data collection routines, evaluating reuse of COTS and OSS components, integrating reuse in software development processes and analyzing own data are major challenges. We observed a great deal of variance among studies regarding the amount of reuse, problem density and productivity gains. It is therefore necessary to have explicit internal goals and baselines, and link benefits to strategic or economic values. It is interesting to notice that only in three studies, the authors were employees of the companies or have stated this relation clearly. The question is whether any feedback was given to industry in the other studies performed by outsides.

In addition to identifying gaps, we provided suggestions for improving reuse research based on the results of the review.

Final comments The evidence is sparse and we may hope that the positive trend of year 2005 with four published papers continues. We found too few empirical studies to generalize findings in several aspects. The review results are presented in several tables that increased the length of the review, but we mean that the data are useful for preparing future research. Performing empirical studies in software engineering does not have a long history, with much to look for and learn about.

Acknowledgement This review was done in the SEVO (Software EVOlution in component-based software engineering) project (SEVO 2006), which is a Norwegian R&D project for 2004–2008 with contract number 159916/V30. We thank Marco Torchiano for his comments on the first version of this review. We also thank the anonymous reviewers for their valuable comments and suggestions.

Appendix A

Table 12 An overview of papers

Reference and publication channel	Objects of study	Type of study
Lim (1994), IEEE Software	1. HP product of 80 KLOC written in Pascal and SPL for manufacturing resource planning. 2. HP product of 64 KLOC written in C for plotters and printers.	Author: Case studies. Review: Experience report, comparing components within each product for quality and data on reuse economics for the two products over years. Agreement: No. Research question and hypotheses are not stated to classify this as a case study.

Table 12 (Continued)

Reference and publication channel	Objects of study	Type of study
Thomas et al. (1997), Journal of Systems and Software	Components from seven medium-scale Ada projects within a narrow domain (simulators at NASA/GSFC Flight Dynamics Division) from a single company. The size of projects is 12.8-27.1 thousand Ada statements or 61–184 KLOC.	Authors:– Review: Case study. Comparing components within a collection of products. Agreement:–
Frakes and Succi (2001), Journal of Systems and Software	Data are from four small sets of C and C ++ modules from four different sources (maximum 16 modules in a set): text processing and retrieval, telecom, telecom and system for medical records.	Authors: Exploratory correlational study as a type of quasi-experiment. Review: Exploratory case study, comparing software modules within four datasets. Agreement: No. Hypotheses are not stated and there is no control by the investigator.
Succi et al. (2001), IEEE Trans. Software Engineering	Data are from two products in the same product family in an Italian medium sized software company that develops accounting software systems. One product (99 KLOC) was developed using the company's standard development cycle (including ad-hoc reuse) while the other product (101 KLOC) was developed after the development of a domain-specific library. Programming language is not given.	Authors: Two-group post-test only experiment. Data collection after completion of the projects. Review: Sister-project case studies (exploratory?). It is not given how the projects were selected. Correlational study. Agreement: No. Assignment of treatments is not discussed, hypotheses are not stated but *t*-test is applied for different means.
Morisio et al. (2002), IEEE Trans. Software Engineering	A single programmer developed five small applications based on an OO-framework (of 10 KLOC in Java and some COTS) and four based on traditional component development. Software sizes were in the range of 256–3,020 LOC and effort per project has been 13–95 person–hours. The applications were network applications such as telelearning and video on demand.	Authors: Exploratory case study or a single-object quasi-experiment. Review: Sister-project case studies. The authors call the study exploratory since one developer is involved, and we also use that. Agreement: Yes.
Mohagheghi et al. (2004), ICSE'04 Proceedings	Data are from three releases of a telecom product of 408–480 KLOC programmed in Erlang, C and Java. This product shares reusable components with a second product in a product family.	Authors: Case study of data mined from industry. Review: Case study. Components from a single product are compared for releases. Agreement: Yes.
Tomer et al. (2004), IEEE Trans. Software Engineering	A model is proposed to compare the cost/effort of different reuse scenarios (e.g., opportunistic versus systematic or new development) and applied on seven software modules in one	Authors: Case study. Review: Example application, comparing reuse scenarios for components where some data are measured and some are estimated.

Table 12 (Continued)

Reference and publication channel	Objects of study	Type of study
	company (electronic systems). Module sizes are not given.	Agreement: No. A model is proposed by the authors and examples of its use are given.
Baldassarre et al. (2005), ICSM'05 Proceedings	Data are from two on-going projects in an Italian SME, one with a reuse-oriented development (ROD) and one without it, otherwise similar in language (COBOL), specification and design technique. Software sizes are not given in the paper. We contacted the first author who said that she would classify the size as medium. The first project developed an application for automatic currency conversion, and the second for managing banking procedures.	Authors: Sister-project case studies, with random allocation of developers to the two products. Review: Sister-project case studies, two on-going projects. It is not given how the projects were selected. Agreement: Yes.
Morad and Kuflik (2005), SwSTE'05 Proceedings	Reuse experiences of initiating a reuse team is described. Two internal reusable assets (a large tool and a small component for electronics industry) and three OSS components are given as examples. Programming languages are C++ and Java.	Authors: Cases. Review: Experience report, similar to Tomer et al. (2004) comparing reuse scenarios for components where some data are measured and some are estimated. Agreement: Yes.
Selby (2005), IEEE Trans. Software Engineering	Data are from 25 software systems from a NASA development environment, ranging from 3 to 112 KLOC of Fortran source code. In addition to analyzing data on project level, data from 2,954 modules with complete data on their development are analyzed to evaluate reusability.	Author:– Review:Case study of data mined from industry, comparing projects related to reuse rate and comparing components in the collection of projects for several attributes. Agreement:–
Zhang and Jarzabek (2005), SPLC'05 Proceedings	A product line was initiated by identifying similarities and differences among four small games (probably 4.5 KLOC totally) in an extractive way and these products were developed again based on the product line architecture (PLA). Further, a new game was developed twice, once based on the PLA and once without it. The J2ME platform (Java) is used for implementation.	Authors: Experiment. Review: Experience report. Replicated product design. Agreement: No. Several characteristics of experiment are missing such as discussion of object or subject selection or hypotheses.

Appendix B

Table 13 Data on reuse approaches

Reference & scale	Development scope	Technical approach and domain scope	Reuse management and initiation	Modification	Reuse rate	Measured unit of reuse	Complementary factors
Frakes and Succi (2001)–S	External reuse of functions is measured.	Compositional (C and C++ files and functions)	Ad-hoc	Verbatim	External Reuse Level/ Frequency/Density (ERL/ERF/ERD) is measured. Mean ERL/ ERF is max 0.47.	Source code (functions)	No.
Morisio et al. (2002)–S	External (developed before experiment inside the company)	OO framework	Systematic in products based on the framework	Verbatim and modified	Around 80% reuse level in the five products based on the OO framework.	Source code of framework as-is	Framework is well documented and easy to learn.
Tomer et al. (2004)–S	Internal	Reuse of software modules	Systematic and controlled	Verbatim and modified	Not given.	Source code in binaries for five assets, Source code for other two	Combination of approaches.
Morad and Kuflik (2005)–S	Internal and external, including an example of OSS	Reuse repository, horizontal and vertical reuse	Systematic	Verbatim	Verbatim reuse 10–15% for internal. Not given for projects using OSS.	Source code in binaries	Management commitment and reuse team but hesitation to enlarge the reuse team, participation of reusers in definition of reusable assets.
Zhang and Jarzabek (2005)–S	Internal	Domain engineering, architecture reuse and meta-components for customization	Systematic, Extractive initiation	Modified by customization	Undefined	Undefined	Extractive and proactive approaches, using experience from an earlier project.
Lim (1994)–M	Internal	Undefined	Systematic (reuse program), Incremental reactive initiation	Undefined	1. 68% reuse 2. 31% reuse	Source code	Incremental (reactive) approach to reuse for case 1, reuse team for case 2.

Table 13 (Continued)

Reference & scale	Development scope	Technical approach and domain scope	Reuse management and initiation	Modification	Reuse rate	Measured unit of reuse	Complementary factors
Succi et al. (2001)–M	Internal	Compositional, call to functions in a general purpose library in one product, and domain and general purpose library in another	Ad-hoc in one product and systematic in another	Seems to be verbatim reuse of functions.	E\RL, ERF and ERD are used for reuse on function level, and are not significantly different in the two products. Mean ERL/ ERF are <0.70.	Source code (functions)	Commitment to improvement.
Baldassarre et al. (2005)–M	Internal	Reuse repository and reuse of templates	Ad-hoc in one product and systematic in another	Verbatim and modified through templates	25% for the product with reuse	Source code	Reuse training.
Thomas et al. (1997)-L	Internal	Architecture reuse and domain library	Systematic	Verbatim and customization (Ada generics in a reuse library that can be instantiated with specific parameters)	Verbatim reuse 4–89% in seven projects. Total reuse (including modified) 31–100%	Source code	Ada language encourages reuse.
Mohagheghi et al. (2004)–L	Internal	Domain engineering, architecture reuse and a defined component model. Vertical and horizontal reuse	Extractive initiation, Systematic	Verbatim	59–61% in three releases of one product in the product family	Source code of components, reused as-is	Management commitment, extractive approach, layered architecture.
Selby (2005)–L	Internal	Architecture and context-dependent module reuse (vertical)	Systematic	Verbatim, slight or major modification	Mean 32% for all the products, 0–82% across the products. Verbatim reuse 0–70%	Source code at module level	Mature reuse process and environment, commitment of developers.

Appendix C

Table 14 Quantitative data presented in the papers and analysis methods

Reference & scale	Data in the paper	Hypotheses	Analysis	Discussion of significance
Frakes and Succi (2001)–S	All data	No (exploratory case study)	Spearman's rank correlation (robust to outliers)	No. Significance of correlation statistics are not tested due to small sample size.
Morisio et al. (2002)–S	All data	Null hypotheses of difference and no alternative hypotheses.	Analysis of covariance to select input variables, least square linear regression to build models.	Significance of variables in the model is discussed.
Tomer et al. (2004)–S	All data	No (example application)	Not applicable.	No.
Morad and Kuflik (2005)–S	Some data and diagrams	No (experience report)	Not applicable.	No.
Zhang and Jarzabek (2005)–S	All data	No (experience report)	Not applicable.	No.
Lim (1994)–M	Mean values of two products over years.	No (experience report)	No.	The results are discussed as significant for HP.
Succi et al. (2001)-M	Mean, max and min, standard deviations, total overview of data.	Not stated but *t*-test is applied for significant different means which means null hypothesis of no difference. No alternative hypotheses.	Normality is discussed. Pearson correlation between pairs, *t*-test for significantly different means.	Correlation is considered significant at 0.05 and 0.01 levels. *P*-values are reported for *t*-tests and no fixed significance level is given, but concluded as significant.
Baldassarre et al. (2005)–M	Box plots with median and scatter plots.	Null hypotheses of no difference and alternative hypotheses of difference.	Mann–Whitney U test for differences between two treatments, Wilcoxon test for successive measures of same treatment.	*P*-values are reported and no significance level is give, but the paper states that there has been a fixed threshold.
Thomas et al. (1997)–L	Mean values, total overview of data and distributions in percentages.	Null hypotheses of no difference or independence. No alternative hypotheses.	Non-parametric Mann-Whitney U test since many parameters were not normally distributed. Chi-square test for testing differences of categorical data.	*P*-values higher than 0.0001 are given and less than 0.01 are considered as significant.

Table 14 (Continued)

Reference & scale	Data in the paper	Hypotheses	Analysis	Discussion of significance
Mohagheghi et al. (2004)–L	Mean, median and standard deviations. Some detailed data, a total overview of data, percentages and scatter plots.	Null hypotheses of no difference and alternative hypotheses of difference. One directional null hypothesis.	Normality is discussed. *T*-test, Mann–Whitney test (when not normal) and Chi-square test are applied.	*P*-values are reported and no fixed significance level is given, but discussed as significant. Effect size and practical significance are discussed.
Selby (2005)–L	Mean, median and standard deviations, scatter plots and histograms.	Null hypotheses of no difference. No alternative hypotheses.	Non-parametric ANOVA using ranked data since several dependent variables were not normally distributed.	Significance level (α) is discussed (e.g. $\alpha < 0.05$ or $\alpha > 0.05$ or $\alpha < 0.001$). *P*-values are not given.

Appendix D

Table 15 Summary of quantitative findings in sister-project studies

Reference & scale	Findings	Discussion of confounding factors
Morisio et al. (2002)–S	Regression model shows that development with an OO framework gives higher apparent and actual productivity than without it (difficult to read how much). Quality measured in less rework improves over time in both cases, but more with a framework than without it.	One single developer was involved but the seven applications were comparable. The cumulative size of software written by a programmer is related to learning and a framework increases conceptual learning and thus productivity. The time sequence of development of applications was a compromise between randomization and the practical needs of the study.
Zhang and Jarzabek (2005)–S	Design quality as measured in OO metrics (complexity, coupling) was almost the same after using a product line architecture compared to before. Size in LOC was reduced by 26.5%. Memory usage decreased from 0.6 to 19% in four games and the games run 2.7–10% faster than before. Effort was reduced by 68% for one game.	The study is replicated product design and one confounding factor is whether the same developers have been involved in replication, which is not answered.

Table 15 (Continued)

Reference & scale	Findings	Discussion of confounding factors
Succi et al. (2001)–M	The product developed with domain library and systematic reuse had 44% less customer complaints per LOC (problem density) than the product developed with ad-hoc reuse. Higher reuse levels in units resulted in lower problem density when a domain library was in place, but not with ad-hoc reuse.	Comparable projects and developers. Size measures are not correlated with customer complain density. The decrease in problem density after delivery is not correlated with the complexity of code.
Baldassarre et al. (2005)–M	MCC of the product developed with systematic reuse was 33% less than the product developed without it, and it had significantly higher apparent productivity (median was almost 38 LOC/person hour compared to 28 in one dataset), while net productivity was not different.	Comparable projects and random assignment of developers.

Appendix E

Table 16 Summary of quantitative findings in component-comparison studies

Reference & scale	Findings	Discussion of confounding factors
Frakes and Succi (2001)–S	Correlational exploratory case study with four datasets: 1. Negative correlation was observed between reuse rate and the number of changes (Deltas) to a module; −0.5 using External Reuse Level (ERL) and −1.0 using External Reuse Frequency (ERF). 2. Positive correlation was observed between reuse rate and subjective quality rating (0.46 Spearman's rank correlation for ERL and 0.62 for ERF). 3. Negative correlation was observed between reuse rate and error density, between reuse rate and Deltas, and between reuse rate and effort per module. Positive correlation was observed between reuse rate and subjective quality rating (the figures for ERF were −0.62, −0.61 and −0.69 and 0.76. For ERL, the correlations were in the same direction but lower). Results regarding apparent productivity were contradictory. 4. Contradictory results regarding apparent productivity.	
Tomer et al. (2004)–S	For all the seven assets, systematic or controlled reuse gave savings relative to new	

Table 16 (Continued)

Reference & scale	Findings	Discussion of confounding factors
	development. For five assets, systematic reuse gave savings between 42 and 81% compared to new development (measured in person–hours). For one asset, controlled reuse was 32% better than systematic reuse, while for others, systematic reuse would be best.	
Morad and Kuflik (2005)–S	Savings of systematic reuse over new development is approximately 50%, and less for other reuse scenarios. For the OSS products, systematic reuse with adaptation would be best, although it is difficult to read exact savings from diagrams.	
Lim (1994)–M	First product: Defect density was 0.9 for reused and 4.1 for non-reused code. Using reused code in combination with new code resulted in 51% reduction in defect density compared to new code and 57% increase in productivity. Second product: Defect density was 0.4 for reused and 1.7 for non-reused code. Using reused code in combination with new code resulted in 24% reduction in defect density compared to new code and 40% increase in productivity. Increased reuse rate (also with modification) is related positively to increase in apparent productivity.	
Thomas et al. (1997)–L	Reuse through slight modification shows 59% reduction in error density and verbatim reuse results in more than 90% reduction in error density compared to new code. Percentage of difficult-to-isolate errors ranges from 12.4% for new components to 14.5% for extensively modified ones. Percentage of difficult-to-complete errors was 22.4% for verbatim reused components compared to 10.1% for new code, which is significantly higher. Components reused verbatim had an average of 24.5 Ada statements and 1.1 withs per component (a measure of external dependency), while new components had an average of 45.8 statements and 3.4 withs per component. Reuse via slight modification shows a 35% reduction in relative rework effort over newly created components, while verbatim reuse provides an 88% reduction. For these modes of reuse, the benefit of fewer errors clearly outweighs the cost of more difficult error correction for reused components.	Smaller components were found to be more defect-prone, but for verbatim reused components, the defect densities were low anyhow. Verbatim reused modules were smaller and had less external dependencies.

Table 16 (Continued)

Reference & scale	Findings	Discussion of confounding factors
Mohagheghi et al. (2004)–L	The number of change requests per KLOC was 0.0012 for reused and 0.0008 for non-reused components; i.e. higher for reused. Using mean values, the fault-density of reused components is 44–61% of the non-reused ones, less difference for modified code. 37% of faults were severe for reused components and 27% for non-reused ones. Reused components were in average 43% modified between two releases, compared to 57% for non-reused ones. The lower fault density of reused components is estimated to reduce the total development effort by 20%.	No relation was observed between size and fault-density. The impact of different programming languages used for reused and non-reused components is rejected since the one used for reused components had more defects per LOC when studied in isolation. Reuse is both vertical and horizontal and thus the impact of differences in functionality is less. Developers of reused and non-reused components were from the same company and had comparable skills.
Selby (2005)–L	On project level, project size was not related to the percentages of modules reused verbatim or with slight modification, but it was significantly positive related to reuse with major modification. On module level, modules reused verbatim had in average 98% less faults than modules newly developed. For modules slightly modified the result was 55%. Fault isolation effort per module was in average 99% less for verbatim reuse and 50% less for modules slightly modified compared to new modules. The number of changes per module was in average 94% less for verbatim reuse and 26% less for modules slightly modified compared to new modules. Modules reused with major modification had the most changes, most changes per LOC and highest change implementation effort (even more than new code). Modules reused verbatim were smaller in size and had less development effort.	Verbatim reused modules were smaller in size and had less input-output parameters.

References

Baldassarre MT, Bianch A, Caivano D, Vissaggio G (2005) An industrial case study on reuse oriented development. In: Proc. 21st IEEE Int'l Conf. on Software Maintenance (ICSM'05), pp 283–292

Basili VR (1990) Viewing maintenance as reuse-oriented software development. IEEE Softw 7(1):19–25

Basili VR, Briand LC, Melo WL (1996) How software reuse influences productivity in object-oriented systems. Commun ACM 39(10):104–116

Birk A, Dingsøyr T, Stålhane T (2002) Postmortem: never leave a project without it. IEEE Softw 19(3):43–45

Boehm B, Brown W, Madachy R, Yang Y (2004) Software product line cycle cost estimation model. In: Proc. 2004 ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE'04), pp 156–164

Conradi R, Li J, Slyngstad OPN, Kampenes VB, Bunse C, Morisio M, Torchiano M (2005) Reflections on conducting an international survey of Software Engineering. In: Proc. 4th International Symposium on Empirical Software Engineering (ISESE'05), pp 214–223
Coverty http://scan.coverity.com/, visited in April 2006
Dedrick J, Gurbaxani V, Kraemer KL (2003) Information technology and economic performance: a critical review of the empirical evidence. ACM Comput Surv 35(1):1–28
Dybå T, Kitchenham BA, Jørgensen M (2005) Evidence-based software engineering for practitioners. IEEE Softw 22(1):58–65
Dybå T, Kampenes VB, Sjøberg D (2006) A systematic review of statistical power in software engineering experiments. Inf Softw Technol 48(8):745–755
Fenton N, Krause P, Neil M (2002) Software measurement: uncertainty and causal modeling. IEEE Softw 19 (4):116–122
Fitzgerald B, Kenny T (2004) Developing an information system infrastructure with open source software. IEEE Softw 21(1):50–55
Frakes WB, Kang K (2005) Software reuse research: status and future. IEEE Trans Softw Eng 31 (7):529–536
Frakes WB, Succi G (2001) An industrial study of reuse, quality and productivity. J Syst Softw 57 (2001):99–106
Frakes WB, Terry C (1996) Software reuse: metrics and models. ACM Comput Surv 28(2):415–435
Gigerenzer G (2004) Mindless statistics. J Socio-Econ 33(2004):587–606
Glass RL (1997) Telling good numbers from bad ones. IEEE Softw 14(4):15–16, 19
Glass RL (2002) In search of meaning (a tale of two words). IEEE Softw 19(4):136, 134–135
Gregor S (2002) A theory of theories in information science. In: Gregor S, Hart D (eds) Information systems foundations: building the theoretical base. Australian National University, Canberra, pp 1–20
Hallsteinsen S, Paci M (eds) (1997) Experiences in software evolution and reuse. Springer
Karlsson E-A (ed) (1995) Software reuse, a holistic approach. John Wiley & Sons
Kitchenham BA (2004) Procedures for performing systematic reviews. Joint technical report, Keele University Technical Report TR/SE-0401 and National ICT Australia Technical Report 0400011T.1
Kitchenham BA, Pickard LM (1998) Evaluating software eng. methods and tools—part 10: designing and running a quantitative case study. ACM Sigsoft Softw Eng Notes 23(3):20–22
Kitchenham BA, Pfleeger SL, Hoaglin DC, El Emam K, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734
Krueger C (2002) Eliminating the adoption barrier. IEEE Softw 19(4):29–31
Lee AS, Baskerville RL (2003) Generalizing generalizability in information systems research. Inf Syst Res 14(3):221–243
Li M, Smidts CS (2003) A ranking of software engineering measures based on expert opinion. IEEE Trans Softw Eng 29(9):811–824
Lim WC (1994) Effect of reuse on quality, productivity and economics. IEEE Softw 11(5):23–30
Lim WC (1996) Reuse economics: a comparison of seventeen models and directions for future research. In: Proc. 4th Int'l Conf. on Software Reuse (ICSR'96), pp 41–50
Madanmohan TR, Dé R (2004) Open source reuse in commercial firms. IEEE Softw 21(6):62–69
Mohagheghi P, Conradi R (2006) Vote-counting for combining quantitative evidence from empirical studies—an example. In: Proc. 5th ACM-IEEE Int'l Symposium on Empirical Software Engineering (ISESE'06), pp 24–26
Mohagheghi P, Conradi R (2007) An empirical investigation of software reuse benefits in a large telecom product. ACM Transactions of Software Engineering Methodology (TOSEM) (in press)
Mohagheghi P, Conradi R, Killi OM, Schwarz H (2004) An empirical study of software reuse vs. defect-density and stability. In: Proc. 26th Int'l Conf. on Software Engineering (ICSE'04), pp 282–292
Mohagheghi P, Conradi R, Børretzen JA (2006) Revisiting the problem of using problem reports for quality assessment. In: Proc. 6th Workshop on Software Quality (WoSQ'06)—as part of Proc. 28th International Conference on Software Engineering & Co-Located Workshops, pp 45–50
Morad S, Kuflik T (2005) Conventional and open source software reuse at Orbotech—an industrial experience. In: Proc. IEEE Int'l Conf. on Software-Science, Technology & Engineering (SwSTE'05), 8 p
Morisio M, Tully C, Ezran M (2000) Diversity in reuse processes. IEEE Softw 17(4):56–63
Morisio M, Romano D, Stamelos I (2002) Quality, productivity, and learning in framework-based development: an exploratory case study. IEEE Trans Softw Eng 28(9):876–888
Norris JS (2004) Mission-critical development with open source software: lessons learned. IEEE Softw 21(1):42–49
Pfleeger SH (1996) When the pursuit of quality destroys value. IEEE Softw 13(3):93–95

Pfleeger SH (2005) Soup or art? The role of evidential force in empirical software engineering. IEEE Softw 22(1):66–73
Pickard LM, Kitchenham BA, Jones PW (1998) Combining empirical results in software engineering. Inf Softw Technol 40(1998):811–821
Ramachandran M, Fleischer W (1996) Design for large scale software reuse: an industrial case study. Proc. 4th Int'l Conf. on Software Reuse (ICSR'96), pp 104–111
Schneidewind NF (1992) Methodology for validating software metrics. IEEE Trans Softw Eng 18(5):410–422
Selby W (2005) Enabling reuse-based software development of large-scale systems. IEEE Trans Softw Eng 31(6):495–510
SEVO (2006) http://www.idi.ntnu.no/grupper/su/sevo/index.html
Shadish WR, Cook TD, Campbell DT (2001) Experimental and quasi-experimental designs for generalized causal inference. Houghton Mifflin Company
Sommerseth M (2006) Component based system development in the Norwegian software industry. NTNU master thesis. http://www.idi.ntnu.no/grupper/su/su-diploma-2006/sommerseth-dipl06.pdf
Succi G, Benedicenti L, Vernazza T (2001) Analysis of the effects of software reuse on customer satisfaction in an RPG environment. IEEE Trans Softw Eng 27(5):473–479
Szyperski C (with Gruntz D, Murer S) (2002) Component software, beyond object-oriented programming, 2nd edn. Addison Wesley
Thomas WM, Delis A, Basili VR (1997) An analysis of errors in a reuse-oriented development environment. J Syst Softw 38(3):211–224
Tomer A, Goldin L, Kuflik T, Kimchi E, Schach SR (2004) Evaluating software reuse alternatives: a model and its application to an industrial case study. IEEE Trans Softw Eng 30(9):601–612
Wang C (1993) Sense and nonsense of statistical inference: controversy, misuse, and subtlety. Marcel Dekker
Webster J, Watson RT (2002) Analyzing the past to prepare for the future: writing a literature review. MIS Quarterly 26(2):xiii–xxiii
Wohlin C, Runeseon P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering. Kluwer
Yin RK (2003) Case study research, design and methods. Sage
Zannier C, Melnik G, Maurer F (2006) On the success of empirical studies in the International Conference on Software Engineering. In: Proc. 28th Int'l Conf. on Software Engineering (ICSE'06), pp 341–350
Zelkowitz MV, Wallace DR (1998) Experimental models for validating technology. IEEE Computer 31(5): 23–31
Zhang W, Jarzabek S (2005) Reuse without compromising performance: industrial experience from RPG software product line for mobile devices. In: Proc. 9th Int'l Software Product Line Conf. (SPLC'05), pp 57–69

Parastoo Mohagheghi is a researcher at SINTEF, Department of Information and Communication Technology (ICT). She received her Ph.D. from the Norwegian University of Science and Technology in 2004 and worked there before joining SINTEF. She has also industry experience from Ericsson in Norway. Her research interests include software quality, model driven development, software reuse, measurement and empirical software engineering. She is a member of IEEE and ACM.

Reidar Conradi received his Ph.D. in Computer Science from the Norwegian University of Science and Technology (NTNU) in 1976. From 1972 to 1975 he worked at SINTEF as a researcher. Since 1975 he has been assistant professor at NTNU and a full professor since 1985. He has participated in many national and EU projects, chaired workshops and conferences, and edited several books. His research interests are in software engineering, object-oriented methods and software reuse, distributed systems, software evolution and configuration management, software quality and software process improvement. He is a member of IEEE Computer Society and ACM.