Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions

L. BRIAND
Carleton University, Canada

E. ARISHOLM
University of Oslo, Norway

S. COUNSELL
Birbeck College, University of London, UK

F. HOUDEK
Daimler-Chrysler AG, Germany

P. THÉVENOD–FOSSE
LAAS–CNRS, France

1. Introduction

Object-Oriented technologies are becoming pervasive in many software development organizations. However, many methods, processes, tools, or notations are being used without thorough evaluation. Empirical studies aim at investigating the performance of such technologies and the quality of the resulting object-oriented (OO) software products. In other words, the goal is to provide a scientific foundation to the engineering of OO software.

This paper summarizes the results of a working group at the Empirical Studies of Software Development and Evolution (ESSDE) workshop in Los Angeles in May 1999. The authors of this paper took part in the working group and have all been involved with various aspects of empirical studies of OO software development. We therefore hope to achieve a good coverage of the current state of the art.

We provide an overview of the existing research and results, future research directions, and important issues regarding the methodology of conducting empirical studies. In Section 2, we cover existing empirical studies and we relate them to the claims usually associated with OO development technologies. Section 3 describes in depth what we believe are important directions for research and, more concretely, precise research questions. Section 4 identifies what we think are important methodological points and strategies to answer these questions.

2. State of the Art

In this section, we first present the current state of knowledge and understanding regarding OO software development. We then describe the most relevant empirical studies published to date. Finally, we summarize what we think are open issues to be addressed by our community.

2.1. Existing Data and Claims

Object-oriented (OO) technologies (programming languages, tools, methods, and processes) are claimed to improve the quality of software product deliverables, to support reuse and reduce the effort of developing and maintaining the software product. However, little evidence exists to support the claims (Jones, 1994; Daly *et al.*, 1996). For example, inheritance and polymorphism are claimed to provide benefits such as greater extensibility and reusability of OO systems (Booch, 1994). However Daly *et al.* (1996) showed results suggesting that, beyond a certain level, inheritance was a serious hindrance to maintainability. Also, the characteristics of OO technologies (such as decentralised architecture, genericity, encapsulation, inheritance, client-server relationships, polymorphism) set a number of new challenges from the perspective of testing. As a result, traditional testing approaches must be revisited to take into account OO mechanisms (Binder, 1996; McGregor, 1997a; Kung *et al.*, 1998). The two basic testing issues, namely, how to define the testing levels (unit, integration) and which test criteria should guide the selection of test data, have to be reconsidered. In addition, the design of test environments could become much more complex due to controllability and observability problems that are drastically increased by encapsulation mechanisms. To summarize, whether or not OO technologies facilitate the testers' work (and then lower testing cost) is still an open issue (Perry and Kaiser, 1990). Furthermore, there is anecdotal evidence that polymorphism and encapsulation decrease testability significantly (Binder, 1994; Voas, 1996).

We can see that there is no simple answer regarding the use and performance of OO technologies. Although they are changing the face of software development, they do not bring the unconditional improvements that were promised or hoped for. From a more general perspective, the software engineering community seems to have followed, once again, the traditions of the past: technology adoption is mostly the result of marketing forces, not scientific evidence.

In addition, many different choices can be made in terms of which processes to follow, which tool support to employ, or which languages and notations to use. It is very likely that various OO technologies will show different properties, advantages and drawbacks. We need to better understand them.

A large amount of work has focused on understanding how the quality of OO artifacts (e.g., design, code) could be assessed. External quality attributes, such as maintainability and reliability, can only be measured late in the software life cycle. We therefore need to identify early indicators of such qualities based, for example, on the structural properties of artifacts. Existing data suggests that there are important relationships between structural attributes and external quality indicators. For example, there is some evidence that some forms of coupling have a negative impact on fault proneness (Briand *et al.*, 1998b; 1999b). However, in general there exists insufficient empirical evidence supporting the usefulness of a vast number of proposed OO measures.

The goals or hypotheses underlying many of the proposed measures are often not clearly stated. The measures are often not defined in a fully operational form (Briand *et al.*, 1999a), making it difficult to replicate the studies by other researchers. Even among the measures that *have* been properly defined and validated empirically, external factors (such as architectural design, developer experience, tools, software engineering processes, organizational maturity, etc) may limit the external validity of the results (Briand *et al.*, 1999b). Consequently, work is required to replicate existing studies in different development environments. This would allow us to better understand how to build quality models under a variety of circumstances, i.e., understand what relationships between early indicators and external quality attributes are stable and what makes them vary across environments.

Some early investigations on testing approaches for OO software have already been performed and published. Each of them presents only partial solutions by placing emphasis on a specific issue, e.g., determination of the testing levels (Jorgensen and Erickson, 1994; Kung *et al.*, 1995; Labiche *et al.*, 1999), reuse of test data based on inheritance (Harrold *et al.*, 1992), test selection strategies for small OO software (Barbey *et al.*, 1996; Thévenod–Fosse and Waeselynck, 1997; Kung *et al.*, 1998), test environments (McGregor, 1997b; Waeselynck and Thévenod–Fosse, 1999). Yet, these partial solutions have not been compared (when they address similar issues), and no attempt to combine them has been made.

2.2. Overview of Existing Empirical Studies

Certainly, much work remains before the impact of object-orientation can be fully understood. Section 2.1 suggests that the state of practice needs improvement. Do we have support for the claims that object-oriented technologies improve the quality of software product deliverables, support reuse, and reduce the effort to develop and maintain the software product? Which of the available technologies are more likely to yield benefits? How do we assess the quality of OO artifacts early on in the software life cycle? How do we define test plans and design test data early in the life cycle? This section attempts to give an overview of empirical validation methodologies and existing empirical studies that attempted to shed some light on some of these questions.
Existing empirical research of the OO paradigm can be classified according to the following (overlapping) research areas:

- Methodology for formal definition, theoretical, and empirical validation of OO measures.
- Empirical evaluation of OO quality models, e.g., maintainability, testability.
- Empirical evaluation of OO design principles.
- Empirical evaluation of OO technologies.

Measurement Validation Principles

Several authors have suggested that measures should adhere to measurement theoretical principles (Zuse, 1991; Fenton, 1992; Fenton, 1994) as a means of evaluating software measures and to qualify the use of certain statistical techniques (depending on the measurement level of the measures). Briand *et al*. (1996a) argues, however, that a more pragmatic approach is likely to provide the software engineering community with more practical results. Several guidelines and frameworks for theoretical and empirical validation of measures have been proposed (Weyuker, 1988; Briand *et al.*, 1995; Kitchenham *et al.*, 1995; Briand *et al.*, 1996b). These frameworks argue that measures should obey certain fundamental properties, that compound measures must be justified by an explicit theory, and that internal measures should be validated empirically against external (quality) attributes. Methodologies for the definition and construction of OO coupling and cohesion measures are proposed in Churcher and Shepperd (1995), Briand *et al*. (1998a, 1999a).

Empirical Assessment of Object-Oriented Design Principles

Surprisingly few papers exist that empirically compare OO technologies and processes to traditional structured techniques. (Jones, 1994) has identified lack of empirical evidence for several claims related to gains in productivity and quality of OO technologies.

In a study described in Basili *et al*. (1996a), OO reuse techniques were applied to eight medium-sized management information systems, using the waterfall life-cycle model. The study indicated *"significant benefits from reuse in terms of reduced defect density and rework as well as increased productivity"*.

Results described in Briand *et al*. (1997b, 1999c) strongly suggest that quality guidelines based on Coad and Yourdon's design principles have a significant beneficial effect on the maintainability of OO design documents. However, preliminary results showed that there was no strong evidence that OO designs were easier to maintain than structured designs.

In Houdek *et al*. (1999), Houdek *et al.*, conducted an experiment comparing structured and OO methods applied to embedded software systems. The results identified only minor differences in development time and quality of the developed systems.

In Sharble and Cohen (1993), one of the few experiments comparing alternative OO technologies was reported. The authors conducted an experiment where they compared a data-driven and a responsibility-driven design method. Two systems were developed based on the same requirement specification—using the data-driven and the responsibility-driven design method, respectively. Structural attribute measures of the two systems were collected and compared. Based on the measured values, the authors suggested that responsibility-driven design produced higher quality software than data-driven design. However, whether the design measures used in this experiment actually measured "quality" was not empirically validated. In other words, the experiment did not involve any direct measurement of external quality attributes.

Empirical Evaluation of Quality Models for Object-Oriented Software

A large portion of empirical research in the OO research arena has been involved with the development and evaluation of quality models for OO software. The immediate goal of this research is to relate structural attribute measures intended to quantify important characteristics of OO software, such as

- encapsulation,
- polymorphism,
- inheritance,
- coupling and
- cohesion,

to external quality indicators such as

- fault proneness,
- development effort,
- test effort,
- rework effort,
- reusability, and
- maintainability.

The main motivations are to be able to assess quality early on in the software life cycle and to be able to use structural attribute measures as surrogates for external attribute measures. This would greatly facilitate technology assessment and comparisons, e.g., in studies such as (Sharble and Cohen, 1993).

Several measures for OO designs have been proposed (Li and Henry, 1993; Chidamber and Kemerer, 1994; Brito e Abreu, 1995; Briand *et al.*, 1997a; Bieman and Kang, 1998) and validated (Basili *et al.*, 1996b; Brito e Abreu and Melo, 1996; Briand *et al.*, 1998a; Briand *et al.*, 1998b; Harrison *et al.*, 1998a; Harrison *et al.*, 1998b; Briand *et al.*, 1999a; Briand *et al.*, 1999b). For historical reasons, the “CK metrics suite” proposed by Chidamber and Kemerer (Chidamber and Kemerer, 1994) are the most frequently referenced OO-design measures.

In Basili *et al.* (1996b), all CK measures except LCOM (Lack of Cohesion in Methods) seem to be useful for predicting class fault-proneness during high- and low-level design phases. In Chidamber *et al.* (1998), it was found that high values in CBO (Coupling Between Objects) and LCOM were associated with lower productivity, greater rework and greater design effort.

In Daly *et al.* (1996), an experiment was conducted to evaluate the effects of inheritance depth on the maintainability of OO software. Results suggest that systems with approximately three levels of inheritance depth may result in reduced time required to perform maintenance tasks by 20% compared to no use of inheritance. However, results from the same study indicate *decreased* maintainability at five levels of inheritance depth.

Benlarbi and Melo (1999) conclude that polymorphism may increase the fault-proneness of OO software. However, as discussed below, these results should be interpreted with care as fundamental flaws undermine the analysis.

Briand *et al*. (1998b, 1999b) report that highly accurate predictive models of fault-prone classes can be developed based on various dimensions of coupling in OO systems. The same data suggest that current measures of cohesion (including LCOM) are poor indicators of fault-proneness.

Binkley and Schach (1998) collected maintenance data from four development projects written in COBOL, C, C++ and Java, respectively. The results suggested that, *"a significant impediment to maintenance is the level of interaction (or coupling) between modules"*. Modules with low coupling were subjected to less maintenance effort and had fewer maintenance faults and fewer run-time failures.

In summary, OO code and design measures have been used to empirically assess how structural characteristics of OO systems affect developer productivity and product quality. When the internal product measures are properly defined and implemented, and after extensive empirical validation against externally observable quality indicators, the use of the internal product measures may eventually provide a common, validated measurement framework which would allow researchers and industry

- to evaluate and improve the efficiency of OO technologies, and
- to evaluate and improve the quality of the resulting OO software products.

Empirical Assessment of Testability of Object-Oriented Software

While several theoretical studies have already focused on testing OO software, few empirical studies of the feasibility and adequacy of proposed techniques have been conducted.

As regards the issue of defining testing levels, two approaches have been defined and implemented in software tools. Both of them aim at minimizing the number of test stubs needed to conduct dependent class tests. First, the OOTM (Object-Oriented Software Testing and Maintenance) Environment (Kung *et al.*, 1995) implements the test order algorithm defined by Kung *et al.*, The second tool implements a different approach (Labiche *et al.*, 1999). Both approaches are based on a class test model (a digraph) which captures the static dependencies between the classes and their objects. But they exhibit notable differences. For example,

- in the OOTM environment, the class test model is constructed by reverse engineering from C++ programs; the test order algorithm applies to both cyclic and acyclic digraphs, that is, whether or not they are cycles in the static dependencies; it does not take into account the dynamic dependencies between object classes (polymorphism and dynamic binding);
- in the other tool, the class test model is derived from the documents produced early in the development process (e.g., UML documents); the algorithm to define and order

the testing levels assume that there is no cycle in the static dependencies; it takes into account both static and dynamic dependencies between object classes (including cycles due to dynamic dependencies).

The OOTM tool has been used for several case studies, e.g., the well-known Interviews library. As for the other tool, a single case study (from the avionics field) has been conducted so far. Clearly, the two approaches have different limitations that have still to be empirically compared.

To our knowledge, no complete experiment, from object-oriented analysis to program testing has been reported, except for the one in Barbey *et al*. (1998b). The experiment is based on the production cell case study initially defined within the framework of a German project. Starting from the informal specification of the case study (Lewerens and Linder, 1995), an OO version of the controller has been developed (Barbey *et al.*, 1998a) using the Fusion analysis and design method (Coleman *et al.*, 1994), and implemented in Ada 95. The controller consists of concurrent objects linked by client-server relationships. The tests were designed according to two different approaches, statistical testing (Waeselynck and Thévenod–Fosse, 1999) and formal testing (Barbey *et al.*, 1996), thus allowing the authors to analyze the feasibility of each approach, the problems met, and the results. This case study is not representative of all OO applications. In particular, there are few inheritance relations, and only between very basic classes. Yet, it provides a significant example of empirical assessment of the testability of certain types of OO software products. In particular, the need to take into account the testability issues early in the software life cycle is emphasized, regardless of the testing approach.

2.3. Common Problems and Open Issues

One pervasive problem across empirical studies in software engineering is the low quality of some study designs and analyses. This is true regardless of the specific topic addressed by the studies. Even when a correct analysis approach is used, results are often misinterpreted. For example, in Benlarbi and Melo (1999), statistically insignificant results are interpreted as being meaningful and strong conclusions are drawn regarding the impact of polymorphism. In Brito e Abreu and Melo (1996), “shotgun” correlations are applied on a very small sample of projects, thus making the risk of committing errors of Type I very likely and the results hardly interpretable. Such practices would not be accepted in other experimental fields but are being published in software engineering. Other experimental fields (e.g., ICMJE, 1988) have defined analysis and reporting procedures that should be followed for a scientific article to be considered as publishable.

As we are maturing, the issue mentioned above is being addressed but this is not an easy undertaking as it involves changing the way we educate software engineering researchers and, to some extent, the way software engineering research is perceived altogether. To some researchers, software engineering is a mathematical science. To the other end of the spectrum, it is perceived as a social, human-intensive process that must be studied using techniques and approaches developed in social and medical sciences.

Regarding empirical studies of OO artifacts, there is a need to re-focus some of the research:

- Investigate artifacts from earlier states of the life cycle, e.g., requirements, architectural design, etc. in order to build early quality models.
- Stop proposing new product measures but investigate the existing ones, following thorough, rigorous, and complete analysis procedures, within the context of well-designed empirical studies. This should also lead to the identification of relevant measurement needs regarding the quality modeling of OO products.
- Replicate studies across many environments, thus establishing a solid body of empirical knowledge on which to perform meta-analysis and draw more general conclusions.

Additionally, the many OO process, notations, and languages that have been proposed need to be investigated through experiments and field studies. In other fields of engineering, social, or medical research, it is uncommon to propose solutions and technologies without going through a thorough evaluation process. Software engineering should not be the exception. This will be further discussed in the next section.

3. Research Directions

In this section, we discuss directions for further research in the field of empirical studies related to object-oriented software development and evolution. First, we identify, based on our experience, the most important objectives for further research. Then, we provide a non-exhaustive list of open research questions matching those research objectives.

3.1. Objectives

The overall objective of empirical studies of object-oriented technologies and products is to gather tangible evidence about its properties and gain deeper insights into the nature of the object-oriented paradigm and its relationship to other approaches. More precisely, we have identified four interdependent goals.

Identify important productivity and quality factors.

The performance and quality of object-oriented technologies and products may depend on many factors, e.g., training, support tools. It is a prerequisite for valid empirical research to know about these factors and control for them. Without such control, research results can only be questionable in the sense that they may be due to other causes than the ones hypothesized. Relevant factors may be related to human factors (e.g., staff experience and training), development processes (e.g., time pressure, methodology) and the product itself (e.g., class coupling, depth of inheritance hierarchy). However, there may also be dependencies between factors (e.g., staff experience may influence the use of inheritance and therefore the depth of inheritance hierarchy). A lot of the work on object-oriented measures can be seen as contributions to this goal (see Section 2) as it helps understand what product characteristics makes the software fault-prone or expensive to develop.

Evaluate OO technologies:

Alternative OO technologies (e.g., methods, and tools) should be assessed and compared. In order to determine the external validity of the obtained results, the studies should specify very clearly the context in which such an evaluation is performed, i.e., characterizing and controlling influential factors.

Building (quality and productivity) models.

Important factors may be used as independent (explanatory) variables in quality or productivity models. Quality and productivity, regardless of the specific way they are measured, are then the dependent variables of the models. In other words, the relationships between independent and dependent variables can be explored and modeled. The resulting models form an essential input to plan, control, or evaluate processes and products. For example, these models can be used to trigger defect-detection activities on specific parts of a system, support impact analysis during maintenance, or devise design guidelines.

Meta-analysis.

In the current literature, they are many instances of research contributions that do not specify clearly the goal of the research. It is often difficult to draw any useful, tangible conclusion. In addition, the setting of the study, the characteristics of the data collected are most of the time superficially described. These problems limit our capability to generalize conclusions to other settings.

3.2. Research Questions

Although the list of potentially relevant research questions could be endless, we will provide some of the most important ones below. They are based on our experience and the discussions that took place during the workshop. We will group the questions according to the goal structure proposed above.

3.2.1. Identify Important Factors

- One important part of determining influential factors is to perform proper measurement. Several useful measurement frameworks have been provided in the recent past. However, the proposed metrics are mainly related to static aspects of object-oriented systems. Measurement of dynamic attributes (e.g., coupling at run-time) has not yet been considered in depth.

- Measurement of single classes has been investigated to a great extent (see references in Section 2). But typical systems are not built as a collection of single, independent classes but from a collection of class clusters (e.g., when using COTS products). Therefore, measuring only single classes is not sufficient and clusters of classes are of interest as well. To date, it is unclear how to transfer the existing measurement techniques from single classes to class clusters.

- The number of measures that have been proposed for object-oriented products is very large. And yet every conference in this field proposes new ones. At this point, there needs to be a shift of effort from defining new measures to investigate their properties and applications on replicated studies. We need to better understand what these measures are really capturing, whether they are really different, and whether they are useful indicators of quality or productivity properties of interest. The need for new measures will then arise from, and be driven by, the results of such studies.

- The application domain is usually seen as a major factor determining the usefulness of measures or the effectiveness of technologies. Application domains may be defined in different ways but we mean here to characterize the type of functionality delivered, the type of development technologies used, the scale of development, the level of complexity of products and any other characteristic which is typically associated with a domain. Variations across application domains make our task more complicated since it limits the external validity of any empirical study.

3.2.2. Evaluation of Object-Oriented Technologies

- The Unified Modeling Language (Booch *et al.*, 1998) (UML) is now becoming a de-facto standard, it is important to investigate its use more thoroughly. It is, for example, possible that a subset of the notation could be used more efficiently (the UML being a quite large set of notations at the moment). Also, certain parts of the formalism may lead to confusion and need more precise semantics.

- Different tools are available to support UML-based development. These tools need to be evaluated and their prerequisite for successful use must be investigated. Many studies have shown that the success of the use of CASE tools is driven by extraneous factors, e.g., training (Bruckhaus *et al.*, 1996)

- Development processes are also proposed such as the Rational (Jacobson *et al.*, 1999) or OPEN (Graham *et al.*, 1997) processes. Again, the introduction of such processes should be carefully monitored since it is likely that they will require some degree of tailoring in each organization.

3.2.3. Building Quality and Productivity Models

- Most quality models reported in the literature are based on measurement that can only be obtained at late stages, e.g., detailed design or coding. Quality models need to be available earlier in the life cycle in order, for example, to drive inspections and ensure early built-in quality.
- Certain aspects of quality have been the focus of very little research. For example, although it is recognized as an issue in object-oriented development, testability has rarely been addressed. Different aspects of maintenance have also been the object of little attention. For example, the factors that affect design and code comprehension or the ease of impact analysis (Briand *et al.*, 1999d).
- There exist studies trying to relate productivity or cost to product characteristics (Nesi and Querci, 1998; Chidamber *et al.*, 1998). But these studies are scarce and use information that can only be obtained in the late stages of design. We need productivity models that can be used earlier on in the course of development, so that accurate project planning and risk analysis can be performed. This may not be solved only through new measurement models and empirical studies, but also by the selection of a development process that documents requirements, specifications, and design decisions early on.

3.2.4. Meta-level Issues

- Quantitative measurement and analysis has its limits. In order to interpret results coming out of empirical studies, one has to rely on qualitative methods. For example, it is rarely the case that quantitative results are interpreted by performing structured interviews of the study participants, administering a debriefing questionnaire, or organizing feedback sessions with the development teams. Without such techniques, there are usually several ways of interpreting results, which may sometimes be the outcome of phenomena that were not thought of. We have to remember that most studies of OO processes and products are exploratory in nature.
- In order to build a body of evidence from a set of empirical studies, results need to be combined and conclusions need to be generalized. This is the field of meta-analysis, which is well developed in other fields such as medicine. However, to allow for meta-analysis, software engineering studies need to be better reported. Important details are often missing from research papers or case study reports. A typical example is that, although significance levels of the effect of independent variable on dependent variables are usually reported, the size effect is almost systematically missing (Pickard *et al.*, 1998).

4. Strategies and Methodologies

The design and performance of empirical studies is a difficult craft. Many pitfalls may compromise the validity of scientific results. Although general principles and techniques are available to perform empirical studies (e.g., experiments, case studies), each discipline needs to develop its own body of experience and strategies to answer its most pressing research questions. For example, although this is an oversimplification, sociology has focused on the design of surveys, medicine on longitudinal studies, and psychology on controlled experiments. Each strategy reflects the working constraints of the respective field.

In other disciplines than software engineering, many of the more revealing empirical studies have been based on prior studies that either support or refute prior claims. Progress in empirical research comes from questioning past research and learning from past mistakes or insights. In this section, we discuss some of the most important pre-requisites for acceptable empirical work in OO software development.

4.1. Success Factors for Empirical Studies

A large number of empirical studies have been undertaken in the past. They range from identification of OO features that may cause higher fault-rates in software, to a study of the effectiveness of different design documentations. The need for meta-analysis of the results of such studies underlies a number of inherent problems in the way that empirical studies have been carried out. For example, if an experiment is carried out, and the results are interesting, it would be useful for other researchers to replicate that experiment as closely as possible. The research reaches a dead end if this is not possible thereby blocking the path to what may be more interesting and insightful research. Consequently, a number of what could be called *success factors* (broad-based factors) which create the conditions for a successful empirical study can be identified.

4.1.1. Nature of the Data

By its nature, an empirical study requires the collection, dissemination and analysis of data. In many System based OO studies, collection of the data alone is problematic. As an example, consider state-of-the-art in OO cohesion metrics (Briand *et al.*, 1998a). The emphasis on cohesion measures seems to have been based around the distribution and use of instance variables of a class. Hence, a class with a single attribute used in all methods of that class is considered cohesive. Alternatively, a class with ten attributes, each of which is used in only one method is considered lacking in cohesiveness.

> The problem with such proposed metrics is the following: most are theoretically flawed either because they produce meaningless and incomparable values or there are counter-examples which render the metric inadequate. Many that are theoretically sound are unsupported by any tool to aid the collection. Of the remainder, analysis and collection from anything other than small systems is computationally intensive with the problems this brings.

One success factor is therefore to ensure in any empirical study that quality of data collection is maintained. This entails ensuring reliability, completeness and efficiency of the data collected. Furthermore, although no measurement is perfect, it must be clearly justified and its underlying assumptions must be made explicit as to aid the interpretation of results. Since the replication of studies is the key to successful empirical research, it is also crucial that any measurement reported be defined in an operational and unambiguous manner.

4.1.2. Consistent Terminology

A common problem in the software engineering community is the inconsistent use of terminology and notation. A good example of this is in the use of the terms analysis and design for OO development. The two terms are used interchangeably. The empirical studies community is no different in this respect; arguments on the correct approach to, and use of, measurement theory in empirical research have only started to abate recently. There have also been questions raised as to the exact meaning of case-studies vs. experiments, the conditions that must hold for research to be claimed to be based on either, and hence whether the results are credible.

Another example of misuse of terminology in the OO world is in use of the term coupling. In an OO setting, there are many forms of coupling that can arise in systems. Establishing what the different forms of coupling are and establishing which are most harmful are still open research questions, although some work has been done already in this area (Briand *et al.*, 1999).

4.1.3. Nature of the Research

In many cases, empirical research is undertaken without establishing beforehand whether the problem is either worth investigating, is too complex at this stage of knowledge or addresses a different underlying issue. A good example of this is in the study of program complexity, where the goals and the concepts under study have lacked clarity (Briand *et al.*, 1996b). Two important reasons for this were in the confusion regarding the practical software engineering problems to be addressed by the research and in the meaning itself of complexity. Successful empirical research would seem to require a clearly defined problem, the results of which are useful to the community and that addresses the problem head-on.

4.2. Design of Case Studies and Experiments

Many factors need to be considered when designing a case-study or experiment (Basili *et al.*, 1986; Pfleeger, 1995). Careful planning is a pre-requisite for credibility in the results. In Harrison *et al*. (1999), details of an experiment involving forty-eight undergraduate students was described; it illustrates the need for care when undertaking such a task. Features of the experiment were:

1. The experiment on the modifiability and understandability of inheritance in C++ systems was based on a previous experiment carried out by Daly *et al*. (1996).

2. A pilot study with twelve students was carried out initially to identify problems that may have hampered the larger experiment.

3. Random allocation of students was to one of twelve different groups. Each group member was allocated to one of four systems. No *learning effects* took place since each student was allocated to just the one system.

4. Random allocation of students to group ensured that the results were unlikely to suffer from experience bias.

5. Strict limits were placed on the time available for the experiment (carried out in a forty-five minute weekly slot).

6. The materials were made available on-line via the web.

7. The results contradicted the earlier experiment. This was considered interesting and added to the body of knowledge about this particular aspect of OO.

Although just one example of an experiment, a number of overriding features are highlighted:

1. Proper materials, for example, code listings, need to be readily available and tested to ensure that minor problems are ironed out before embarking on the full experiment. The pilot study mentioned revealed several features of the tasks required which could have undermined results of the later experiment (e.g., the wording of the questions).

2. Subjects should know the tasks they have to carry out so as to obtain representative results, e.g., careful training may be a pre-requisite if the subjects are not familiar with the experimental tasks.

3. The groups should be comparable in terms of make-up, experience. This is important as we know individual capability can be an overriding factor in software development. However, this is far to be trivial since we do not know very well how to characterize experience in software engineering.

4. The experiment should be repeatable, and the materials should be made available to a wider community. A *replication package* should be made available containing details of the experiment.

The last point is important for building up a knowledge base of past experiments. One possible offshoot of the work described would be to undertake the same experiment using experienced practitioners. This could reveal differences between student and experienced programmers; a worthwhile area of research. In the OO community, so little is known about many facets. This situation can only improve via sharing of results and resources.

5. Conclusions

There is a large amount of research to be done in terms of studying empirically object-oriented software development and maintenance. Empirical research has, so far, been scarce and exploratory. We are still learning about the right ways to measure the most important attributes of our research. For example, concepts like testability, maintainability or cohesion are still very elusive. In many studies, this kind of measurement is very questionable, thus threatening the validity of the results.

Furthermore, we are still exploring ways to design empirical studies and overall research programs to answer efficiently the most pressing research questions. The investigation of new medical treatments follows various phases where animal experiments, small scale case studies on voluntary patients, carefully sampled field studies and surveys are being sequentially performed so as to limit the risks at each stage. How can we, in our field, investigate new technologies in a way that limit the cost of investigation, the risks for pilot projects, and ensure minimal benefits for software development organizations? This is a question that will take time to answer and that will require substantial experience that we do not have yet. As no real laboratory experiments are possible, it will likely involve student experiments during the early stages of research. If the results of such experiments show promise, this will likely be followed by low risk, small-scale pilot projects, before moving to representative development projects. Such a scheme implies the close collaboration of academic institutions and industry but also the acceptance, like in medicine, that empirical studies are worthwhile investments.

This paper has provided a representative overview of what is our current understanding as a community and has identified pressing research issues and directions. Considering the pace of technology change in software engineering, it is however evident that such a program requires a great deal more resources that are currently being spent. As a community of research, we therefore need to grow to undertake the challenges we are facing. In addition, more efficient schemes of collaboration with industry, as well as the betterment of our education in empirical methods, will also be key success factors.

References

Barbey, S., Buchs, D., and Péraire, C. 1996. A theory of specification-based testing for object-oriented software. *Proc. 2nd European Dependable Computing Conference (EDCC-2)*, Taormina, Italy 303–320.

Basili, V. R., Selby, R. W., and Hutchens, D. H. 1986. Experimentation in software engineering. *IEEE Transactions on Software Engineering* 12(7): 733–743.

Basili, V., Briand, L., and Melo, W. 1996a. How reuse influences productivity in object-oriented systems. *Communications of the ACM* 39(10): 104–116.

Basili, V. R., Briand, L. C., and Melo, W. L. 1996b. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(10): 751–761.

Barbey, S., Buchs, D., and Peraire, C. 1998a. Modelling the production cell case study using the fusion method. Technical Report 98/298, EPFL-DI.

Barbey, S., Buchs, D., Gaudel, M-C., Marre, B., Péraire, C., Thévenod–Fosse, P., and Waeselynck, H. 1998b. From requirements to tests via object-oriented design. *DeVa Year 3 Deliverables* 331–383. Also available as LAAS Report 98476.

Benlarbi, S., and Melo, W. L. 1999. Polymorphism measures for early risk prediction" *21st International Conference of Software Engineering (ICSE'99)*, Los Angeles, CA, 334–344

Bieman, J. M., and Kang, B. K. 1998. Measuring design-level cohesion. *IEEE Transactions on Software Engineering* 24(2): 111–124.

Binder, R. 1994. Design for testability in object-oriented systems. *Communications of the ACM* 37(9): 87–101. Also available in Kung *et al.*, 1998).

Binder, R. 1996. Testing object-oriented software: a survey. *Software Testing, Verification & Reliability* 6(3/4): 125–252.

Binkley, A. B., and Schach, S. R. 1998. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. *20th International Conference on Software Engineering (ICSE'98)* 452–455.

Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummins Publishing Company Inc.

Booch, G., Rumbaugh, J., and Jacobson, I. 1998. *The Unified Modeling Language User Guide*. Addison-Wesley.

Briand, L., Daly, J., and Wust, J. 1998a. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering* 3(1): 65–117.

Briand, L., Devanbu, P., and Melo, W. L. 1997a. An investigation into coupling measures for C++. *19th International Conference on Software Engineering (ICSE'97)*. Boston, 412–421

Briand, L., El Emam, K., and Morasca, S. 1995. Theoretical and Empirical Validation of Software Product Measures. ISERN Technical Report 95-03.

Briand, L., Emam, K. E., and Morasca, S. 1996a. On the application of measurement theory in software engineering. *Empirical Software Engineering* 1(1): 61–68.

Briand, L., Morasca, S., and Basili, V. R. 1996b. Property-based software engineering measurement. *IEEE Transactions on Software Engineering* 22(1): 68–85.

Briand, L. C., Bunse, C., Daly, J. W., and Differding, C. 1997b. An experimental comparison of the maintainability of object-oriented and structured design documents. *Empirical Software Engineering* 2(3): 291–312.

Briand, L. C., Daly, J. W., Porter, V., and Wust, J. 1998b. A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems. Technical Report ISERN-98-07.

Briand, L. C., Daly, J. W., and Wust, J. 1999a. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering* 25(1): 91–121.

Briand, L. C., Wust, J., Ikonomovski, S. V., and Lounis, H. 1999b. Investigating quality factors in object-oriented designs: an industrial case study. *21st International Conference of Software Engineering (ICSE'99)*. Los Angeles, CA, 345–354.

Briand, L. C., Bunse, C., and Daly, J. 1999c. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. Fraunhofer Institute for Experimental Software Engineering, Germany, ISERN-99-07. To be published in *IEEE Transactions on Software Engineering*.

Briand, L. C., Wuest, J., and Lounis, H., 1999d. Using coupling measurement for impact analysis in object-oriented systems. Fraunhofer Institute for Experimental Software Engineering, Germany, ISERN-99-03. To be published in the proceedings of *IEEE ICSM'99*. Oxford, England.

Brito e Abreu, F. 1995. The MOOD metrics set. *Proc. ECOOP'95 Workshop on Metrics*.

Brito e Abreu, F., and Melo, W. 1996. Evaluating the impact of object-oriented design on software quality. *Proceedings of the Third International Software Metrics Symposium (METRICS'96)*. Berlin.

Bruckhaus, T., Madhavji, N., Janssen, I., and Henshaw, J. 1996. The impact of tools on software productivity. *IEEE Software.* 13(5): IEEE Computer Society Press.

Chidamber, S. R., Darcy, D. P., and Kemerer, C. F. 1998. Managerial use of metrics for object-oriented software: an exploratory analysis. *IEEE Transactions on Software Engineering* 24(8): 629–637.

Chidamber, S. R., and Kemerer, C. F. 1994. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering* 20(6): 476–493.

Churcher, N. I., and Shepperd, M. J. 1995. Towards a conceptual framework for object-oriented software metrics. *Software Engineering Notes* 20(2): 69–76.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. 1994. *Object-Oriented Development: The Fusion Method*. Object-Oriented Series, Prentice Hall.

Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1(2): 109–132.

Fenton, N. 1992. When a software measure is not a measure. *Software Engineering Journal* 357–362.

Fenton, N. 1994. Software measurement: a necessary scientific basis. *IEEE Transactions on Software Engineering* 20(3): 199–206.

Graham, I., Henderson-Sellers, B., and Younessi, H. 1997. *The OPEN Process Specification*. Addison-Wesley.

Harrison, R., Counsell, S. J., and Nithi, R. V. 1998a. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering* 3(3): 255–273.

Harrison, R., Counsell, S., and Nithi, R. 1999. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Proceedings of Empirical Assessment in Software Engineering (EASE 99)*. Keele, Staffordshire.

Harrison, R., Counsell, S., and Reuben, V. 1998b. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering* 24(6): 491–496.

Harrold, M-J., McGregor, J. D., and Fitzpatrick, K. 1992. Incremental testing of object-oriented class structures. *Proc. 14th IEEE Int. Conference on Software Engineering (ICSE-14)*. Melbourne, Australia, 68–80. Also available in (Kung *et al.*, 1998).

Houdek, F., Ernst, D., and Schwinn, T. 1999. Comparing structured and object-oriented methods for embedded systems: a controlled experiment. *ICSE'99 Workshop on Empirical Studies of Software Development and Evolution (ESSDE)*. Los Angeles, 75–79.

International Committee of Medical Journal Editors 1988. Uniform requirements for manuscripts submitted to biomedical journals. *Ann. Inter. Medicine* 1988(108): 258–265.

Jacobson, 1., Booch, G., and Rumbaugh, J. 1999. *The Unified Software Development Process*. Addison-Wesley.

Jones, C. 1994. Gaps in the object-oriented paradigm. *IEEE Computer* 27(6): 90–91.

Jorgensen, P., and Erickson, C. 1994. Object-oriented integration testing. *Communications of the ACM* 37(9): 30–38. Also available in (Kung *et al.*, 1998).

Kitchenham, B. A., Fenton, N., and Pfleeger, S. L. 1995. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering* 21(12): 929–944.

Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C., Kim, Y-S., and Song, Y-K. 1995. Developping an object-oriented software testing and maintenance environment. *Communications of the ACM* 38(10): 75–87. Also available in (Kung *et al.*, 1998).

Kung, D., Hsia, P., and Gao, J. (eds.) 1998. *Testing Object-Oriented Software*. IEEE Computer Society.

Labiche, Y., Thévenod–Fosse, P., Waeselynck, H., Durand, M-H., and Farail, P. 1999. A test ordering tool for object-oriented programs. *Proc. 10th European Workshop on Dependable Computing (EWDC-10)*. Vienna, Austria, 57–61.

Lewerens, C., and Linder, T. (eds.) 1995. Formal development of reactive systems: case study production cell. Lecture Notes in Computer Science, Vol. 891, Springer-Verlag.

Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. *Journal of Systems and Software* 23(2): 111–122.

McGregor, J. D. 1997a. Quality assurance. *Journal of Object-Oriented Programming*. Monthly columns since January 1997.

McGregor, J. D. 1997b. Parallel architecture for component testing. *Journal of Object-Oriented Programming* 10(2): 10–14.

Nesi, P., and Querci 1998. Effort estimation and prediction of object-oriented systems. *Journal of Systems and Software* 42: 89–102.

Perry, D. E., and Kaiser, G. E. 1990. Adequate testing and object-oriented programming. *Journal of Object-Oriented Programming* 2(5):13–19. Also available in (Kung *et al.*, 1998).

Pfleeger, S. L. 1995. Experimental design and analysis in software engineering. *Annals of Software Engineering* 1: 219–253.

Pickard, L., Kitchenham, B., and Jones, P., Combining software engineering results in software engineering. *Proceedings of the EASE'98 Conference*. Keele, UK.

Sharble, R. C., and Cohen, S. S. 1993. The object-oriented brewery: a comparison of two object oriented development methods". *Software Engineering Notes* 18(2): 60–73.

Thévenod–Fosse, P., and Waeselynck, H. 1997. Towards a statistical approach to testing object-oriented programs. *Proc. 27th Symposium on Fault-Tolerant Computing (FTCS-27)*. Seattle, 99–108.

Voas, J. 1996. Object-oriented software testability. *Proc. 3rd International Conference on Achieving Quality in Software*. Chapman & Hall, 279–290.

Waeselynck, H., and Thévenod–Fosse, P. 1999. A case study in statistical testing of reusable concurrent objects. *Proc. 3rd European Dependable Computing Conference (EDCC-3)*. Prague, Czech Republic, to appear in September.

Weyuker, E. J. 1988. Evaluating software complexity measures. *IEEE Transactions on Software Engineering* 14(9): 1357–1365.

Zuse, H. 1991. *Software Complexity: Measures and Methods*. de Gruyter.