Empirical Software Engineering, 2, 381–403 (1997)
© 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Process Modelling and Empirical Studies of Software Evolution (PMESSE'97)
Workshop Report

R. HARRISON
Department of Electronics and Computer Science, University of Southampton, UK

L. BRIAND AND J. DALY
Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany

M. KELLNER
Software Engineering Institute, CMU, USA

D. M. RAFFO
School of Business Administration, Portland State University, USA

M. J. SHEPPERD
Department of Computing, Bournemouth University, UK

Abstract. Much progress is being made in both the areas of process modelling and software metrics. However, neither of these concepts is complete without the other: processes cannot be improved if no assessment of quality is available, and metrics are useless if they cannot be applied in order to assess the evolution of systems. The PMESSE (Process Modelling and Empirical Studies of Software Evolution) Workshop, held in Boston MA, on May 18, 1997, brought together researchers and practitioners from both of these fields, and stimulated some very lively debate on these issues. This collection of reports reflects the work done by the Workshops five Working Groups.

Dialogue between the software metrics and process modelling communities is essential. This workshop succeeded in bringing together researchers with a wide range of research interests, and the resulting discussions were very animated. Perhaps the most contentious point of discussion lay in the 'level of granularity' question; some suggested that only high-level measurements were sensible, whereas others preferred to 'divide and conquer' the system, providing detailed process measurement schemas. No doubt this debate will continue.

Keywords: empirical studies, maintenance, legacy systems, metrics

1. Empirical Business Process Modelling

1.1. Acknowledgements

The objective of this working group was to identify how models and metrics are used together in real software development settings. The group also discussed how models and metrics support each other in solving practical problems which are of interest to the software community—industry and academics. To this end, we identified the purposes of models and metrics, enumerated categories of modeling approaches and itemized different types of metrics. We chose to focus on software processes, although many of the conclusions generalize to a broad variety of business processes. The group member were David Raffo, Marc Kellner, Mikio Aoyama (Niigata Institute of Technology, Japan), Maurizio Morisio (Politecnico di Torino, Italy), Keith Phalp (University of Southampton, UK) and Juan Ramil (Imperial College, UK).

1.2. Position

Work in the process modelling arena has been engaged in developing a number of languages and approaches for modelling software development processes. These languages and approaches have different strengths and weaknesses. They have been applied to various aspects of the software development process focusing on process steps and work flows, organizational hierarchies and communication channels, agents and activities, tasks and work products and many other elements. Table 1 taken from Curtis et al. (1992) shows a number of different modelling languages and some of the perspectives which they capture. The different perspectives are defined as follows:

- Functional represents the process elements that are being performed and the information flows or entities which are relevant to those process elements.
- Behavioral represents the process elements that are performed, their sequence of execution, as well as how they are performed through feedback loops, iteration, complex decision-making conditions and so fourth.
- Organizational represents where and by whom (which agents) in the organization the process elements are performed.
- Informational represents the informational entities produced or manipulated by the process including data, artifacts, products (intermediate and final).

However, many of these approaches fail to represent quantitative data and therefore are not suitable for predicting process performance or supporting many practical decisions related to process management. In short, these models are very limited in their usefulness (primarily limited to supporting process understanding or automating routine portions of the process). The usefulness of these models for management and improvement activities can be greatly enhanced by including various metrics which are consistent with the purpose of the model.

Many metrics have been developed. Some have been more useful than others for providing status (one of the possible purposes of metrics and models) of the software development effort—such as defects detected in design inspections, effort expended to date, schedule remaining to date, among others. Alternative uses of metrics might include providing predictive information (another common purpose of metrics and models) and supporting decisions to modify an action plan based on the metric value. One example of this is the use of internal and external design metrics (Zage et al., 1994) which identify "high risk" modules and provide guidance on characteristics to be achieved in redesigning.

However, metrics need to be organized into some integrated package and carefully defined in order to provide useful information and results. If they are not, a company may find that

Table 1. Perspectives for process information and applicable language bases. (Source Curtis et al., 1992).

Language Bases	Functional	Behavioral	Organizational	Informational
Procedural programming languages	X	X	0	X
Systems analysis and design	X	0	X	X
AI languages and approaches	X	X	0	0
Events and triggers	0	X	0	0
State transition and petri-nets	X	X	X	0
Control flow	0	X	0	0
Functional languages	X	0	0	0
Formal languages	X	0	0	0
Data modeling	0	0	0	X
Object modeling	0	0	X	X
Precedence networks	0	X	0	0
Quantitative modeling	X	X	0	0

Legend: X indicates that the language base captures the perspective, 0 indicates that the language base does not capture the perspective (Curtis et al., 1992)

it collects a great deal of information regarding their software development process which may not be useful.

The GQM paradigm (Basili and Rombach, 1988) has given the software community the concept that meaningful metrics must support a well defined goal. This working group proposes that models of software development (process models and others) must also be created to support a specific goal or purpose. Moreover, these models help to structure and organize the use of metrics. This concept is illustrated in figure 1 which shows the Purpose, Question, Model, Metric (PQMM) paradigm. Here the purpose of the model both motivates and categorizes questions that management is interested in learning about. These questions, in turn, direct the formulation of a model which can be used to help answer the questions. Concurrently, metrics are also defined to support the model and to address the questions posed. These metrics are defined in terms of the model's elements and are thus coupled to the process being modelled. A list of commonly observed purposes are listed in table 2. A partial list of potential model perspectives, types, and techniques are listed in table 3. A non-exhaustive list of metrics categories with examples are listed in table 4. The point is that software development models and metrics are best used in an integrated fashion and can support many different purposes. Models and metrics are mutually supportive and their coordinated use provides a synergistic effect.

It is the experience of the working group members that in the real-world, project managers are often interested in achieving multiple purposes and have a number of questions which they would like to address. Model types and techniques are usually better suited to some purposes and questions than to others. For example, a regression model would be good for quantitative prediction but would not be a strong candidate for supporting process understanding. Similarly, a given metric type may fit better with some model types and techniques than with other. For example, the metric of employee turnover rate fits well with

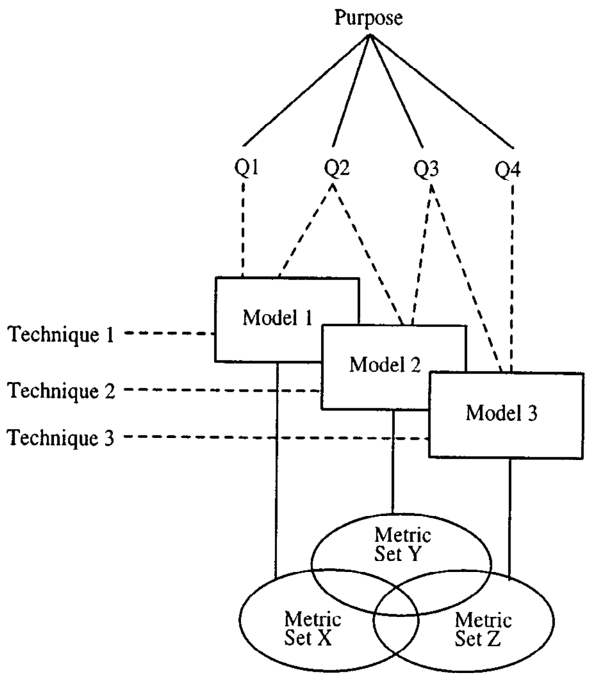


Figure 1. PQMM paradigm.

a simulation model but is difficult to incorporate with a data flow model. This situation often drives the need for integrated models which require a complex set of metrics.

1.3. Summary

Models and metrics have always been inextricably intertwined—implicitly or explicitly. Models and metrics are both most effective when developed to achieve a well defined purpose. There are a variety of practical purposes which project managers are interested in accomplishing. When used together for a well defined purpose, models and metrics provide a synergistic combination which increases the value of the results and expands the scope of the problems that can be addressed by a given model or metrics paradigm.

2. Studying Large Legacy Systems

2.1. Acknowledgements

This section summarises the output of the working group tasked with presenting, reviewing, critiquing and comparing alternative methods of studying large legacy software systems.

Table 2. A non-exhaustive list of common purposes motivating the definition of software development models and metrics.

Purposes	Explanation
Understanding	Supports process understanding. Examples include: Process Tasks, Organizational Structure,Roles and Agents
Prediction	Predicts characteristics of process or product downstream Examples include: Operational Characteristics such as: Number of Defects, Schedule, Effort by Phase, Tasks Completed
Planning	Examples include: Policy Assessment, Process Alternatives, Risk Analysis Staffing and Scheduling
Replanning	Predicts likely outcome based on updated data. For example, replanning based on updated project status.
Control	Examples include: Project Status, Process Guidance
Process Enactment	Automation of the process environment.
Process Benchmarking	Evaluating processes from other companies and other internal sources.
Process Improvement	Predicting the impact of process changes
Optimization of the Process	Examining alternative processes to determine best potential choice.
Hypothesis and Theory Testing	Conducting controlled experiments

Table 3. Model categories (non-overlapping dimensions).

Model Perspectives (from Curtis et al., 1992)	Behavioral Organizational Functional Informational
Modelling Types and Techniques	Simulation (e.g. Discrete Event, Systems Dynamics) Process Flow (e.g.Data Flow Diagrams, Statecharts, Flow Charts) Statistical (e.g. Time Series Analysis, Regression, Statistical Process Control) Analytic (e.g. COCOMO, SLIM)

The focus was upon empirical methods. We concluded that the study of such systems is important, that there is a shortage of such work, that the existence of configuration management (CM) systems and defect tracking systems offer a largely untapped opportunity and that quantitative and qualitative techniques are complementary. The group members were Martin Shepperd, Mark Harman (University of North London, UK), Chris Lott (Bellcore,

Table 4. Metrics types.

Product	Derived from characteristics of the products (e.g. size, complexity, number of modules, etc.)
Process	Based on properties of the process, including tasks, work products, or other operational characteristics (effort, schedule, etc.)
Organizational	Based on organizational properties such as structure, communication channels, and others
Model Based	Derived from properties intrinsic to the model and used as an indicator of process goodness (e.g. Connectivity metrics (Phalp and Counsell, 1997))

USA), Adam Porter (University of Maryland, USA) and Larry Votta (Bell Laboratories, USA) to whom recognition and thanks must be given.

2.2. Introduction

Despite pioneering work over twenty years ago (e.g., Lehman and Belady, 1976) there are still comparatively few published studies of large legacy systems and, indeed, there is much about the behaviour of such systems that we still do not understand. This is surprising given the preponderance of such systems and their economic importance to many organisations. Although there is no exact definition of what constitutes a large legacy system, in practice we believe they are easy to recognise and are characterised by having many components and many people who have worked on them. Such systems have evolved over time and have value to the stakeholders. Typical examples are the software systems for controlling telephone switches. These systems comprise millions of lines of code and are often well in excess of ten years old. Many new builds are made per year. There are also problems with the heterogeneity of the software and/or the hardware.

It is important to stress that maintenance is in fact a multi-faceted activity. We identify at least three perspectives.

1. Making individual modifications (understand the existing system, evaluate proposed changes, implement and validate).
   - design recovery,
   - program comprehension,
   - impact analysis,
   - regression testing.
2. Coping with evolution (the nature of change and how to limit its impact).
   - structured programming,
   - object-oriented programming languages,
   - software architectures,

- configuration management.

3. Managing the maintenance process (maintenance is costly and somewhat unpredictable).

   - develop metrics and models
   - predict when components are likely to change
   - estimate cost

For this reason, we need to be clear whose perspective we are adopting. Empirical studies that do not make this clear and that have no focus, can be difficult to interpret and their results even more difficult to apply. The combination of measurement and process modelling techniques can be useful in this regard. Likewise, the use of methods such as GQM (Basili and Rombach, 1988) which encourage those carrying out the study to be explicit about their objectives and viewpoint.

This section goes onto consider why we might wish to study large legacy systems and what we might hope to learn. Next we briefly consider barriers to such work. We then turn to evaluate the data sources available and analysis methods open to us. We conclude by restating our view that there is an untapped opportunity in studying such systems.

2.3. Motivation

The main reason to study legacy systems is in order to answer questions. These fall into two categories, (i) those concerned with understanding the past and the present and (ii) those concerned with predicting the future. Below we list some questions for which it would be extremely useful to start to obtain insights, if not precise answers.

(i) Understanding maintenance and legacy systems

- Do systems decay (i.e., do legacy systems become more expensive as they get older?)
- What causes a system to decay? - at code level: various programming language issues - at programming language level - at architecture level - at process level: e.g., turnover - at environment level
- How do (a) code, (b) documentation and (c) requirements evolve/change over time?
- What strategies are adopted for coping with change? (E.g., designing code for change)
- How resistant is the code to change? There are two possible meanings: - What are the consequences of a change to a module; do changes ripple across the whole system? - Given that the environment/requirements change(s), how easy is it to make the change?
- Is there a stable and consistent maintenance process?

(ii) Predicting future behaviour:

- What modules are likely to be buggy, changed in future releases, expensive to test?
- How much effort will a given maintenance change consume?
- What will be the future maintainability of a system?

In general, one needs understanding (i.e. answers to type (i) questions) in order to be able to construct prediction systems (i.e. answer type (ii) questions). Note also that even the understanding questions divide into those that adopt a "snapshot" view of a system or process and those that take a longer term view and are based upon trends.

2.4. Problems

Given the economic significance of legacy systems and our lack of detailed understanding of such systems, we need to consider those inhibitors that have impeded the empirical study of such systems.

The first inhibitor is the effect of scale. The very nature of such systems is that they are very large and this is a dominant factor upon their behaviour. This has tended to rather restrict the value of conducting small scale laboratory experiments. It also makes the data collection extremely onerous without automated support. In the past such support has seldom been available, or where it has, often only the most trivial metrics can be collected. Moreover, the use of multiple languages and platforms substantially compounds the problem.

The second inhibitor is the time dimension. There can be considerable problems associated with collecting data over long periods of time. Past data can be lost, forgotten or of doubtful provenance. Maintenance processes and data recording procedures can change over time, rendering trend analyses suspect. On the other hand, if we reject past data, waiting for future data can impose unacceptably long delays upon studies. Organisations would like results sooner rather than later.

The third inhibitor is the problem of many sources of variation. Large maintenance projects are extremely complex. One has to contend with variation from the maintenance changes, the underlying product, the maintenance processes, technology and personnel. This list is probably not exhaustive. It can be difficult to separate or analyse the effects of different factors.

The fourth inhibitor is the problem of external validity. Which principles apply across multiple sites? How can we generalise, especially from case study evidence? If we can't generalise from case studies except into theory (Yin, 1984) how can we help software maintainers through this type of study? What does it mean to replicate a study, particularly given the enormous variation between sites? How do you link abstract theories and concrete aspects of systems, i.e., if you have a principle from one system, how can it be applied to the next situation?

The fifth inhibitor is that many phenomena are best suited to economic analysis. A legacy system is a resource that has economic value to an organisation. Decisions are made on the basis of financial analysis. If we restrict our view to a technical one, there will be many aspects of software maintenance that we will not be able to adequately explain.

The final inhibitors are the software organisations themselves. In order to conduct any empirical study, one needs access to the phenomenon being studied. Clearly, this has time implications upon associated maintenance staff. Pressures of work can militate against this being made available. Confidentiality is another problem. Ideally, results from studies should be published, however, some organisations are reluctant, even if the data is sanitised. Moreover, the less contextual information provided, the harder the findings are to interpret and the harder the study is to replicate.

2.5. Techniques

Firstly, we consider different data sources. These might include:

- static/dynamic analysis of code
- version control or CM systems (code again)
- other documents (architecture, high-level design, requirements)
- data such as plan/actuals (project plan versus execution)
- various compliance documents (e.g., ISO 9000)
- inspection and testing reports
- accounting/time reporting systems
- financial data of the business unit
- email archive
- minutes of project meetings
- results of risk analysis
- user’s views/recollections
- “walking dictionaries” for qualitative data
- process observation

Much of this work might be likened to forensic software engineering.

Data derived from the study of legacy systems typically exhibits certain distinctive characteristics. The data sets are very large which although creating certain problems, also offers almost unique opportunities within the realms of software engineering data analysis. Especially when one considers that in many other studies a dataset of 15 or 20 points is the norm. Significant amounts of qualitative data is another characteristic. This is very important and can provide a vital context within which to interpret quantitative data. Moreover , many important aspects of maintenance may not be amenable to direct measurement (e.g., management style). The datasets are inherently multi-variate. There may be many, and very subtle, interactions between variables. Skewing of the data and the existence of extreme outliers is another problem. For this reason statistical tests that make assumptions of normality should be used with caution. Lastly, CM systems can allow retrospective analysis by rolling back to earlier versions of a software system.

Recall that we differentiated between questions relating to understanding and those relating to prediction. This is an important distinction since the former is exploratory in nature whilst the latter is using a given model to generate outputs concerning the future. These tend to call for rather different techniques. Analysis methods for understanding include:

- Documenting system evolution
- Longitudinal vs snapshot analyses
- Data visualisation
- Data mining techniques (e.g. artificial neural nets (ANNs), rule induction, clustering)
- Ethnography
- Advanced static analysis

Methods for building prediction systems include:

- Documenting system evolution
- Statistical (e.g. regression analysis and discriminant analysis)
- Rule based
- Machine learning (e.g. rule induction, regression tree, ANNs and case based reasoning).

Where large datasets exist the machine learning techniques might be particularly attractive since large training sets will be available.

2.6. Summary

We have argued that large legacy systems are an under studied area of software engineering. Despite some of the problems outlined, there is an opportunity to deploy new and powerful analytic techniques such as data mining, due to the potentially very large datasets. We believe qualitative and quantitative analyses are complementary. Likewise measurement and process modelling.

An important area for further consideration is methodological: how may we generalise from our findings, particularly when conducting case studies? Related is the issue of replication. It is important that the community take this seriously and that we do not have to rely upon the isolated findings of single teams studying single phenomena.

We would certainly urge caution about generic “maintenance metrics” and encourage workers to differentiate between the validation of metrics (do they properly represent the attribute being measured?) and the validation of prediction systems (what is the relationship between predicted and actual values?).

To summarise, we believe there are exciting opportunities that will enable the software engineering community to significantly advance its understanding of large legacy systems.

3. Measurement of Object-Oriented Systems and its Applications

3.1. Acknowledgements

This short report summarizes and builds on the results of the ”Object-Oriented Measurement and Applications” group discussion. The group aimed at better understanding the state of the art and the issues measurement researchers and practitioners face with the shift to object-orientation. The groups members were Lionel Briand, Prem Devanbu (AT&T Research Labs), Lutz Prechelt, and Barbara Unger (both from the University of Karlsruhe, Germany).

3.2. Introduction

With the change of development paradigm to Object-Orientation (OO), our ways of measuring software development products and processes are expected to change. New activities are likely to take place during development (e.g., the reuse and instantiation of design patterns) and products such as design documents will show changes in structure and information content (e.g., inheritance hierarchies). Many articles exist on the topic of measurement of object-oriented design and code. What have we learnt so far? What are the caveats in the current research activities and outputs? What important issues are still to be investigated? These are the questions that are briefly addressed in this report.

3.3. What Have We Learnt?

More than a hundred papers propose, discuss, or assess measures defined in the context of the OO paradigm (see Briand et al., 1996a, Briand et al., 1997a) for structured reviews of the literature). Obviously, this is a surprisingly rich body of work considering that most of these articles have been published within the last five years. Measures have been proposed for attributes such as cohesion, coupling, inheritance hierarchy, size, and complexity. Many of these measures show common properties and similarities. However, many of them are based on different experimental hypotheses about what makes an OO design/program more difficult to develop or maintain, thus leading to higher error rates and effort. As described in Briand et al. (1996a) and Briand et al. (1997a), many of the differences among measures are related to a limited number of decision criteria. At a high level, some examples of criteria related to coupling are

- What dependencies, between classes or class objects, contribute to coupling? Examples are aggregation relationships, method parameters and return types, method invocations.

- The unit of analysis or measurement domain, i.e., the level of detail at which information is gathered. Measures can be defined at the method, class, set of classes, or system level, although many of them are defined at the class level since this is the basic component of the OO paradigm.
- Other important decision criteria are: the procedure used to count dependencies, whether indirect dependencies are taken into account, the way inherited class members are handled, the direction of dependencies.

A first lesson learnt is that there are many ways to measure attributes such as the ones mentioned above. The difficulty is that all of them may make sense, in various contexts and for various purposes. A corollary is that there is little consensus among researchers about what should be the underlying experimental hypotheses of such measures, e.g., which class dependencies constitute class coupling? The main reason is that people have different experiences and backgrounds, resulting in different hypotheses and hence measures. The realm of object-oriented measurement is a very large one to investigate and no simple, a priori reduction in scope may be applied.

Another related lesson learned is that, not only have many different measures been proposed for any given attribute, but the meaning itself of these attributes seems to be subject to wide interpretation in the literature. Several authors (Briand et al., 1996b, Weyuker, 1988) have been proposing axiomatic frameworks in order to constrain the meaning of attributes such as coupling, cohesion, or complexity. But no wide consensus exists with that respect.

Some of the proposed measures show a relationship with class fault proneness or maintainability. For example, some of the Chidamber and Kemerer measures (Chidamber and Kemerer, 1994) as well as the coupling measures proposed by Briand et al. (1997b, 1994) have been empirically validated and have shown to be useful early quality indicators (Basili et al., 1996). Daly et al. (1996) assessed, in a controlled setting, the impact of inheritance depth on maintainability. Similarly, Dvorak (1994) showed that conceptual inconsistency (referred to as entropy) is likely to increase as we travel down an inheritance hierarchy. However, empirical investigations are still scarce and very few of the proposed measures have been shown to be useful to quality modeling.

Last, in order for the measures to be operationally defined, it is often important to make sure that language specifics are clearly handled by the definition. For example, C++ friend classes, the possible use of multiple inheritance (C++ but not Smalltalk), the different mechanisms for inheritance, e.g., whether the inherited private part of a class is also accessible or not.

3.4. Caveats

One simple issue to address is that measures are not always precisely and operationally defined. That is, their usage requires further interpretation from the practitioners before being collected, e.g., should inherited attributes and methods be taken into account? This is of course a hindrance to progress since researchers and practitioners implicitly refer to different measures using the same labels. This issue is particularly crucial when one tries to replicate experimental results.

This is due in part to a lack of common terminology, definition framework, and formalism. Even comparing measures is, in most cases, a difficult exercise. As a result, most measures are used without critical review and comparison, their weaknesses are not well identified, and, as a result, an optimal and practical use of the current state of the art is extremely difficult. This may explain in part the lack of empirical investigation discussed in the next paragraph.

Maybe the most important issue, if we are to make progress regarding the definition of valid, useful measures for object-oriented design and code, is the lack of empirical studies. As mentioned earlier, for most of the defined measures, there is no evidence that the measures are meaningful, useful, or that their underlying assumptions are supported. Another related problem is that some of the existing empirical studies appear to be incomplete or inadequate to provide a high degree of confidence in the results, e.g., no distribution or outlier reporting and analysis. Since there is no clear standard about how such analyses should be reported, this problem is likely to linger on. There is certainly much to be learnt from other fields with this respect, e.g., medicine, social sciences.

3.5. Outstanding Issues

If we as a community are to be able to exchange precise ideas and arguments, then the basic concepts we have to live with, such as cohesion, coupling, or complexity, have to be defined as precisely as possible. It is therefore important that, as a community, we work towards building definition frameworks. One rigorous way to do it is through the definition of axiomatic frameworks, specifying what necessary mathematical properties a measure should fulfill in order to measure a given attribute. Several such frameworks have been proposed in the literature and they are mostly complementary (Briand et al., 1996b, Weyuker, 1988). However, such research activities are still too rare for our community to make progress at a sufficient pace.

Assuming the attributes we investigate are clearly defined and that measures are reasonably thought to measure these attributes, then the relationships between these measures and quality aspects of interests must be identified and quantified (Briand et al., 1995), e.g., fault-proneness, maintainability. This is necessary in order to show that these measures are useful indicators of one or several aspects of product quality. Depending on the nature of the data, various modeling techniques can be used to do so, such as least-square regression, logistic regression, or machine learning approaches such as classification trees. In addition, various data analysis techniques aimed at performing factor analysis (e.g., principal component analysis) and outlier analysis (e.g., Mahalanobis distance) should be typically found in such studies. One difficulty is the measurement of the quality aspects of interests. How does one go about measuring maintainability? How to ensure a reasonable construct validity? First, such concepts are likely to be multidimensional and therefore require several measures to be fully captured, e.g., measures of understandability, impact analysis difficulty, modifiability, and testability. Second, the measures to be used are likely to be based on simplifying assumptions that may hold in a given organization but that bears, a priori, no external validity, e.g., all software components are consistently documented and documentation is therefore not a confounding factor affecting modification effort in the organization under study.

Once measures are validated and are shown to be meaningful quality indicators, then comes the issues related to the construction of useful and usable quality models for architectures, designs, and complete systems. This is discussed in the next section.

3.6. Applications of Measurement for Object-Oriented Systems

There is a broad range of applications for measurement in the context of OO system development and maintenance. Measurement must be perceived as an instrument to help achieve better quality and productivity, not as an end in itself. One application of measurement in this context is to assess the complexity of OO designs and thereby identify their fault-prone parts (Briand et al., 1997b) and the likely cost of subsequent development or maintenance activities (Chidamber et al., 1996). Even earlier in the development process, some measurement may be used, in combination with architecture assessment methodologies such as SAAM (Kazman et al., 1995), to assess and compare architecture alternatives. Measurement-based quality models may also be used to drive verification and validation. For example, fault-prone classes or subsystems may be the focus point of inspection and testing activities. Also, regardless of the OO design method employed, one needs to devise design guidelines in order to fulfill pre-determined acceptance design criteria. In order to do so, the field study of actual development processes and products is necessary and the measurement of development products is a prerequisite to perform it. In addition, measurement help define objective quality criteria which may be verified automatically through static analyzers, a very important issue for large systems. Measurement may also help assess the impact of strategies such as the use of design patterns (Prechelt et al., 1997) on the products, and hence on their quality. Although there is encouraging evidence, the adequacy and feasibility of using measurement in the applications mentioned above mostly remains to be investigated. Integrating measurement into higher level, practical application methodologies remains one of the main challenges we have to face.

4. Empirical Studies of Large Software Systems

4.1. Acknowledgements

The aim for this working group was to present, review, critique and compare alternative methods of studying large software systems, and of evaluating process improvement. This report summarizes and builds upon the group discussion that took place with respect to software measurement in general and the strengths and weaknesses of performing observational studies, pre-experiment studies, quasi-experiments, and controlled experiments to achieve these goals.

Five people participated in the working group. The members were John Daly, Jonathan Cook (New Mexico State University, USA), Luigi Lavazza (Politecnico di Milano, Italy),

Manny Lehman (Imperial College, UK), and Torsten Nelson (University of Waterloo, Canada).

4.2. Introduction

Measurement is an essential activity for improvement: measurement allows us to quantify current practice, provide the basis for improvement from current practice, and then quantify the improvements we set out to achieve. Software measurement can be used for the following activities (all of which are either directly associated with improvement or leading to improvement): to characterize, monitor, predict, control, evaluate, and change aspects of software products and software development and maintenance processes. In the context of this working group the discussion was mainly concerned with characterisation, control, and evaluation. These different measurement activities allow us to characterize the development process used or characterize the software with respect to some interesting external attribute, identify relationships that influence the performance of the process or the software technology, and compare and assess the utility of different processes and software technologies. Just as the different goals of measurement were discussed, so were the different types of empirical study that can be conducted.

4.3. Empirical Study Techniques

It was agreed that we can conduct empirical research through measurement by performing any one of four different types of empirical study (this is not definitive given that surveys are also a type of empirical study from which measurement can be performed, but this technique was not discussed). However, it was apparent that differences exist in the terminology used for the types of empirical study available—shown by the fact that agreement could not be achieved on definitions. The definitions used here are the ones presented in Daly et al. (1997) (where the authors hope these definitions will serve as a starting point to help reach a consensus about terminology).

Observational study: In an observational study, the investigator does not expose subjects to any treatment. Data may be collected during the study, e.g., through visual observation, interviews, and data collection forms, or may be retrospective as when data from a measurement database or historical records are used. Observational studies are almost always conducted in the field or using field data, except perhaps when the unit of analysis is the individual programmer.

Pre-experiment study: In a pre-experimental study, a treatment is applied to only one-case (an organisation, a project, a single group of subjects, a single subject, etc.). A comparison is performed against a baseline and the study is always conducted in the field. This is usually conducted as an investigation of a new technology, e.g., configuration management or inspections, introduced into an organisation's environment to see how useful it is when compared to current practice.

Quasi-experiment: In a quasi-experiment there is no random assignment of subjects to treatments (i.e., non-equivalent groups are likely), one or more treatments, e.g., different types of development methodology, may be applied to two or more groups, direct comparison of groups is performed, and they are performed in either field or lab settings (although most likely in field settings which presents constraints regarding the level of control available). Depending on the constraints, a quasi-experiment can be either conducted in representative conditions where the level of control is minimal (like a pre-experiment except more than one case is examined) or conducted in more controllable conditions of a laboratory setting (like a controlled experiment except random assignment cannot be employed).

Controlled experiment: A controlled laboratory experiment takes place in a setting constructed by the investigators in order to conduct the study. In the controlled experiment, random assignment of subjects is performed, there is an explicit experimental design with one or more treatments to two or more groups, and direct comparison amongst groups is performed. All variables thought to confound the results should be held constant across all the subjects. A high level of control can be achieved, but at the expense of being able to conduct the study in a field setting.

Strengths and weaknesses of these techniques are discussed in detail in Daly et al. (1997) with respect to the following attributes: construct validity, internal validity, external validity, ease of replication, potential for theory generation, potential for theory confirmation, and cost per subject of study.

In the context of large software systems and evaluating process improvement it was clear the most important aspect of an empirical study was external validity (the degree to which the results of the study can be generalised), i.e., producing results based on ‘real world’ data. From a manager’s perspective is was decided that unless this was true the impact of the empirical results would be minimal—consequently, observational and pre-experimental studies seem to be the most relevant types of studies. However, it was also argued that the laboratory is a useful place to begin an investigation for several reasons. First, it allows us to increase our understanding of a particular phenomenon which can lead to a refinement of our hypotheses as well as provide the opportunity to refine our measures, measurement tools and procedures, and experimental materials. Second, the results of quasi-experiments and controlled experiments can be used to convince managers to perform measurement and investigation in their industrial setting, e.g., witness the body of empirical research performed on software inspections in both the laboratory and the field. A multi-method approach, i.e., combining different types of studies, therefore was argued to be the best approach to increase understanding of a particular phenomenon.

4.4. Discussion

What was also clear among the participants is that measurement in general and empirical investigation has to be goal-oriented. If it is not understood why these activities are being performed then the results or findings are unlikely to be of much use. It was also decided that the following activities should be considered when studying large software systems and evaluating process improvement.

- Be very precise in your goals of measurement—use explicit goal based mechanisms, e.g., GQM, to state the aims of your study. If the measurement goals are not stated clearly in the beginning then your measurement activities will have a large chance of failure.
- Start any measurement activities small and build them up incrementally. Also, always look to provide something tangible from the first measurement activity. This should impress the need for measurement in the organisation and should provide future buy-in.
- Feedback any findings as early as possible to encourage people to continue to measure and give them confidence that measurement is not a 'black hole' activity, i.e., they provide additional effort for which they get nothing in return.
- Be cost effective. There is no point trying to measure or evaluate everything right at the beginning. Not only will this be both expensive and obtrusive, it will increase the chance of failure.
- Finally, consider the use of existing data—it may be very useful as a beginning point of reference. However, this should still be approached in a goal-oriented manner, i.e., state the questions you are interested in answering before examining any existing data. Do not simply proceed to analyse for potential relationships without any motivation or reasoning behind your analysis. One argument was that perhaps the best possible use for existing data is to attempt to establish baselines with it. Of course, one must also be concerned with how correct the existing data is.

4.5. Summary

It was argued that any type of empirical investigation must be goal-oriented—without explicit goals the study is unlikely to be of use. In the context of large software systems and evaluating process improvement, the study should be concerned with collecting and analyzing real world data—considered by managers to be the most important aspect of any study. On the other hand, studies in the laboratory can be useful 'first-steps' with which to convince and encourage further empirical investigation in the field.

5. Finding Ways to Work Together

5.1. Acknowledgements

This report reflects the discussion between the Working Group members, who were Rachel Harrison, Linda Ott (Michigan Technical University, USA), Antony Powell (University of York, UK), Carolyn Seaman (University of Maryland, USA), and Richard Webby (IESE, Kaiserslautern and University of New South Wales, Australia).

5.2. Goals and Objectives

The goals of this working group were to consider alternative approaches to empirical evaluation, and ways of crossing the researcher/practitioner divide. How can we transfer research results and ensure that the research which industry needs is done?

In more detail, our specific objectives were as follows:

1. Identify obstacles preventing collaboration and propose ways of overcoming these
2. Identify ‘good’ collaborative models
3. Identify criteria for successful collaboration

The following sections discuss each objective in turn.

5.2.1. Obstacles

Secrecy prevents some employers from making their process models explicit. A solution may lie in re-assuring companies that academics are prepared to sign non-disclosure agreements, and in ensuring that intellectual property rights are safe-guarded. Co-authorship of papers can be used as inducement.

Focus - researchers are not always pragmatic, and so do not always focus on the right questions. Academics need to be prepared to compromise in order to find the right level of abstraction and correct focus for their work.

Insufficient rewards - industrial collaboration is time consuming, which may deter researchers who are pressurised to publish frequently. Funding and tenure committees should be made aware of the importance of collecting real-wold data.

Reputation and respectability - the reputation of a University Department can affect whether or not industrialists will be ready to collaborate. The image of academics as people uninterested in pragmatic, practical matters is another problem. However, empiricism is gaining respect, partly through the popularity of models such as CMM and SPICE. Also, we have a number of international centres of excellence (such as the SEL, Maryland, and the Fraunhofer IESE) which are helping to raise the standards of empirical software engineering research.

Rate of change - of technology can present problems. Academics sometimes become embroiled in detail. The use of laboratory kits, containing the materials used for experiments, would enable others to repeat experiments in different domains, so providing meta-level analyses.

Communications - terminology and specialised notation can cause communication barriers. Technological issues are easier to address than social ones. Academics should be encouraged to spend time on-site in industry; frequently those who do so are given few rewards for their efforts.

Funding - projects are often cancelled at short notice, before commencement or completion. Start-up costs for research projects can be very high. However, some companies will use a percentage of their profits for research.

Research/Development divide - the leap from blue-sky research to the application or take-up of research ideas by industry can be too great. The wide variety of application domains means that experiments must be repeated in a number of different domains before conclusions can be drawn. We need to maintain a balance between pure and applied research.

Education - students are unable to appreciate the problems involved in large-scale software engineering. Universities should encourage students to go into industry, preferably through sandwich courses, or part-time MScs. We should teach transferable skills, rather than particular technological solutions. The University of New South Wales in Australia has set a good example of collaboration, by launching a 4 year degree centred around the Personal Software Process, during which students spend at least 3 months in industry.

5.2.2. Good Collaborative Models

We identified 3 different research models:

1. The Rolls-Royce model, in which a company instigates and directs the research, funding a researcher to spend time on-site intermittently for three years.

2. The CAS (IBM Centre for Advanced Studies) model, which funds Research Fellows over 3 years, enabling academics to spend up to 4 months per year at CAS.

3. The Fraunhofer model, in which dedicated researchers spend two-thirds of their time on industrial projects, the rest being spent on research.

5.2.3. Criteria for Successful Collaboration

By a ‘successful’ collaboration, we mean long-term continuous collaboration which addresses the needs of both the researcher (primarily a contribution to the field and provision of empirical evidence suitable for publication) and the needs of industry (primarily process improvement).

1. Good communication links are extremely important.

2. Clear objectives must be established and agreed from the start.

3. Industry-driven collaboration has the greatest chance of success.

4. A good business case is essential.

5.3. Summary

Research in the laboratory is a necessary first step, as it can improve experimental design and improve the probability of asking the right questions; some research questions can be addressed very well in this way. However, there is still a need to encourage industrialists to regard collaboration with academics as beneficial to business, and to encourage academics to address the questions which are of importance and interest to industry.

References

Basili, V. R., Briand, L., and Melo, W. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(1): 751–761.

Basili, V. R. and Rombach, H. D. 1988. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14(6): 758–773.

Briand, L., Daly, J., and J, W. 1996. A unified framework for coupling measurement in object-oriented systems. Technical Report 14, Fraunhofer Institute for Experimental Software Engineering, Germany.

Briand, L., Daly, J., and J, W. 1997a. A unified framework for cohesion measurement in object-oriented systems. Technical Report 05, Fraunhofer Institute for Experimental Software Engineering, Germany.

Briand, L., Devanbu, P., and Melo, W. 1997b. An investigation into coupling measures for c++. In *Proc. ICSE 97*.

Briand, L., El Emam, K., and Morasca, S. 1995. Theoretical and empirical validation of software product measures. Technical Report 03, Fraunhofer Institute or Experimental Software Engineering, Germany.

Briand, L., Morasca, S., and Basili, V. R. 1994. Defining and validating high-level design metrics. Technical Report CS-TR 3301, University of Maryland.

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

Chidamber, S., Darcy, D., and Kemerer, C. 1996. Managerial use of object-oriented software metrics. Technical Report 750, University of Pittsburgh Katz Graduate School of Business.

Chidamber, S. R. and Kemerer, C. F. 1994. A metric suite for object oriented design. *IEEE Transactions on Software Engineering* 20(6): 467–493.

Curtis, B., Kellner, M. I., and Over, J. 1992. Process modeling. *Communications of the ACM* 35(9): 75–90.

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, An International Journal* 1(2): 109–132.

Daly, J., El Emam, K., and Miller, J. 1997. An empirical research methodology for software process improvement. Technical Report ISERN-97-04, Fraunhofer Institute for Experimental Software Engineering, Germany.

Dvorak, J. 1994. Conceptual entropy and its effect on class hierarchies. *IEEE Computer*.

Kazman, R., Abowd, G., Bass, L., and Clements, P. 1995. Scenario-based analysis of software architecture. Technical Report GIT-CC-95-41.

Lehman, M. M. and Belady, L. A. 1976. A model of large system development. *IBM Syst. J.* 15(3): 225–252.

Phalp, K. and Counsell, S. 1997. Counts and heuristics for the analysis of static models. In *Proceedings of ICSE'97 Workshop on Process Modeling and Empirical Studies of Software Evolution*.

Prechelt, L., Unger, B., and Philipsen, M. 1997. Documenting design patterns in code eases program maintenance. In *Proceedings of ICSE'97 Workshop on Process Modeling and Empirical Studies of Software Evolution*.

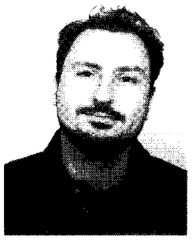
Weyuker, E. J. 1988. Evaluating software complexity measures. *IEEE Transactions on Software Engineering* 14(9): 1357–65.

Yin, R. K. 1984. *Case Study Research—Design and Methods*. SAGE Publications.

Zage, W. M., Zage, D., and Wilburn, C. 1994. Achieving software quality through design metrics analysis. In *Proceedings of the Twelfth Annual Pacific Northwest Software Quality Conference*, Portland, OR.

Rachel Harrison is a member of the Engineering Faculty at the University of Southampton. She obtained an MA in Mathematics from Oxford University, an MSc in Computer Science from London University, and a PhD in Computer Science from the University of Southampton (1991). Her current research interests center around empirical software engineering, particularly measurement and modeling of the object-oriented paradigm, the evaluation of formal methods and of hypermedia systems.

Dr. Harrison is a member of the IEEE Computer Society, the ACM, and the BCS.

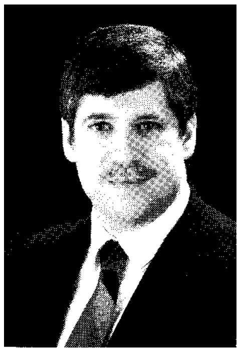


Lionel C. Briand received the B.S. degree in geophysics and the M.S. degree in Computer Science from the University of Paris VI, France, in 1985 and 1987, respectively. He received the Ph.D. degree, with high honors, in Computer Science from the University of Paris XI, France, in 1994.

Lionel started his career as a software engineer at CISI Ingénierie, France. Between 1989 and 1994, he was a research scientist at the NASA Software Engineering Laboratory, a research consortium: NASA Goddard Space Flight Center, University of Maryland, and Computer Science Corporation. He then held the position of lead researcher of the software engineering group at CRIM, the Computer Research Institute of Montreal, Canada. He is currently the head of the Quality and Process Engineering Department at the Fraunhofer Institute for Experimental Software Engineering, an industry-oriented research center located in Rheinland-Pfalz, a beautiful wine region of Germany. His current research interests and industrial activities include measurement and modeling of software development products and processes, software quality assurance, domain specific architectures, reuse, and reengineering.

John W. Daly received the B.Sc. and Ph.D. degrees in Computer Science from the University of Strathclyde, Glasgow, Scotland in 1992 and 1996, respectively. He is currently a software engineering researcher in the Fraunhofer Institute (FhG IESE), Kaiserslautern, Germany where he has been employed since April, 1996.

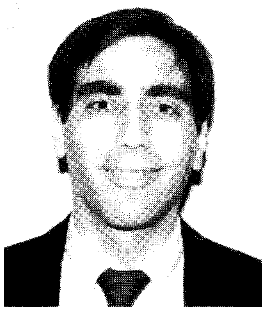
John's current research interests and industrial activities include software measurement and software process improvement, conducting empirical research by means of quantitative and qualitative methods, and object-oriented development techniques.



Dr. Marc I. Kellner is employed as a senior scientist at the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, Pennsylvania, USA. Kellner has pioneered and led much of the research on software process modeling conducted at the SEI, and has published more than two dozen papers on software process issues.

Prior to joining the SEI, Kellner was a professor at Carnegie Mellon University, where he established and directed a degree program in Information Systems. He has also served on the faculty of The University of Texas (Austin). He received his Ph.D. in Systems Sciences (specializing in MIS) from the Graduate School of Industrial Administration at Carnegie Mellon University. He also holds B.S. degrees in Physics and in Mathematics, and M.S. degrees in Computational Physics and in Systems Sciences, all from Carnegie Mellon.

Kellner's research interests include software processes, software process modeling and definition, quality management for software, and software maintenance.



D. M. Raffo is an assistant professor of Operations Management and Information Systems in the School of Business Administration at Portland State University. He received his Ph.D., MME, and MSIA degrees from Carnegie Mellon University. Dr. Raffo's dissertation work and current research are in the area of quantitative software process modeling and prediction, business process re-engineering applied to software firms, empirical software engineering, and simulation. This work was conducted in conjunction with the Software Engineering Institute (SEI) and IBM. He has also received grants from the National Science Foundation and the Software Engineering Research Center (SERC). Dr. Raffo is a member of IEEE Computer Society, the ACM, and INFORMS.

M. J. Shepperd