

Experimentation in software engineering: A new framework

W-E.A. Mohamed^a, C.J. Sadler^a, D. Law^b

^a*School of Computing, University of North London, London N7 8EA, UK*

^b*The National Computing Centre, Oxford Road, Manchester, M1 7ED, UK*

ABSTRACT

The absence of a systematic and flexible evaluation methodology, that can characterise the effect of methods and tools on the development process, has made any process improvement program a difficult task. The DESMET project aims to develop and validate an evaluation methodology which will quantify the effect of the use of development methods and tools.

DESMET (Determining an Evaluation Methodology for Software Methods and Tools) is a project partly funded by the DTI (IEDSATP/1150) and SERC (GR/F 38556). The collaborators in the project are the National Computing Centre, GEC-Marconi Software Systems, Racal Research Ltd and The University of North London.

The first phase of the project has critically surveyed current evaluation approaches which were classified as 'quantitative' or 'qualitative'. It was concluded that DESMET must support both types and provide guidelines to identify which method is appropriate to the particular circumstances of the user. To achieve this, the design of DESMET was based on a multi-module architecture. The paper outlines the architecture of DESMET and the rationale behind it.

Formal experimentation is one of the quantitative techniques that DESMET will support. The paper presents a new framework for conducting formal experiments in a software engineering environment. The paper will also discuss how managerial and sociological issues can affect formal experimentation in software engineering.

418 Software Quality Management

INTRODUCTION

Formal (Controlled) Experimentation has always been a powerful technique in the advance of traditional sciences; attempts to employ this in software engineering are scant. A recent study, Quantum [16], has concluded that hypothesis testing and controlled experiments should form an integral part of any software engineering research programme that attempts to investigate attributes for assessing or understanding the relationship between the development process and product quality. The DESMET project, at its early stages, recognised the importance of formal experimentation and in particular its use in evaluating the effect of software methods and tools on process productivity and product quality, for more details see DESMET [5]. Although, DESMET proposes other courses of action in many circumstances, a special module has been designated in order to provide a new framework with practical guidelines for implementing formal experimentation, when this is appropriate, in software engineering environments. This paper represents the current state of this framework and its underpinning philosophy.

FORMAL EXPERIMENTATION

The term Formal Experimentation (or Experimentation), in this paper, refers to the rigorous application of statistical principles (Replication, Randomisation and Blocking) to facilitate comparisons of the effect of several factors on a process or a product.

In Software Engineering Formal Experimentation aims mainly at comparing the effect of different software development methods and/or tools on the development process or the developed product e.g. Card, McGarry and Page [4]. The method/tool under investigation is called the treatment, in statistical terminology the effects on aspects of the process or the product are measured by means of the response variables. The experiment is run according to a pre-defined plan which arises out of the design of the experiment.

One important characteristic of formal experimentation is that factors, called state variables, which may affect the results of the experiments (by influencing the values of the response variables) are formally controlled.

The existence of a causal relation between the treatment and the response variables is postulated in a statement called the hypothesis. Formal experiments are primarily employed for testing hypotheses and in an attempt to establish the confidence with which we may assert the effect of the treatment.

The use of formal experiments implies that some kind of comparison will be carried out. There are two main types of comparisons: simple comparison between two treatments and the comparison of many treatments.

OVERALL ARCHITECTURE OF THE DESMET METHODOLOGY

Initial methodology design has taken place from the point of view of the end-user of the methodology. This has led to the concept of a 'user interface module' (Known as EMS) and a number of procedural modules, which contain the technical methods of evaluation which DESMET provides and supporting procedures. EMS (Evaluation Method Selection) will help the user to decide which combination of the procedural modules should be used for a particular evaluation.

In later versions, the procedural modules may not all need to be visible to the users of the methodology, some being hidden behind EMS. Conversely, others will be directly accessible for use by 'expert' users. The modules do not, with one exception, have software support and vary greatly in nature, content and development effort, more details in Law and Naeem [11].

Seen from the developers' point of view, the modules of the methodology fall into four functional groups as follows:

Group one: Modules concerned with setting up an evaluation project

This group has four modules. The first is of the nature of a decision support system and attempts to elicit a number of factors from the user which influence the choice of an appropriate evaluation approach and hence the procedural modules to be 'invoked'.

The second module deals with the problems of analysis and justification of investment in methods and tools. It will also help to assess whether a specific evaluation exercise is economically worthwhile.

The third module aims to provide an objective means of establishing the ability of an organisation both to use and benefit from using the DESMET methodology.

A particularly important conclusion from the first phase of (DESMET [5]) was that the DESMET methodology cannot concentrate solely on the technical and accountancy issues of evaluation. The fourth module of this group includes advice for organisations on managerial and sociological issues associated with establishing evaluation programmes. The framework, presented in this paper, highlights some of the issues included in this module.

Group two: Modules concerned with quantitative methods

These modules will form a technically coherent set, all being statistically based. Three of the modules are concerned with the way in which evaluations can be organised i.e. formal experiments, case studies and surveys. The fourth module

420 Software Quality Management

is concerned with a specific technique that can be used to measure software productivity, i.e. cost models. Where appropriate they use concepts, techniques and terminology of the discipline of experimental design.

Group three: Modules concerned with qualitative methods

This group consists of two modules. The first will attempt to define current best practice of feature analysis. The second will aim to be a bridge between feature analysis and quantitative methods by allowing the direct estimation of the likely effects of the use of methods and tools on quality and productivity.

Group four: The Measurement System modules

One common thread underlying all quantitatively-based evaluation is the need for a valid method of interpreting results i.e. a "baseline" against which productivity and quality changes can be assessed. In this group, The Data Collection and Metrication (DCM) Module will as one of its main components provide guidelines for establishing a measurement baseline. The other module in this group is a Data Collection and Storage Subsystem. It is a PC-based software package that provides valuable practical support to the DCM module.

PROBLEMS OF EXPERIMENTATION IN SOFTWARE ENGINEERING

There are four main problems that face formal experimentation in the software industry. These are cost, controlling differences in developers' abilities, the effect of development methods and tools used in part of the development process on other parts and the evolutionary nature of software development environments e.g. Schach [18].

Prohibitive Cost

Since formal experimentation is based on comparisons, the usual approach to setting up an experiment, for example, to evaluate a number of software development tools is to form two or more groups all developing exactly the same system but each using one of the tools under evaluation. The cost of replicating the development effort of medium-to-large systems twice or more would be prohibitive in a commercial environment. There are few organisations that can make a commitment to formal experimentation on purely technical grounds. Much of the work, therefore, has occurred in academic institutions whose interest is less likely to be informed by operational imperatives, more details in Mohamed & Sadler [13].

Wide Variations in Developers' Abilities

The major part of software development is labour intensive. The differences in the developers' abilities and motivation can be dramatic as they are affected by many factors such as training, experience, ability to work in teams, academic background and age. To minimise the effect of individual differences on the final results of an experiment, large numbers of replicates (teams) may have to be chosen. This may not be possible in a small organisation and in many cases

could be financially unfeasible. Randomisation can help in factoring out the problem of individual differences by allocating developers to teams at random in the hope of averaging out differences in abilities. However, sometimes the differences in abilities may be so varied that randomisation alone would not be compensation enough.

Multi-Phase Effect of Development Methods and Tools

The effect of a particular development method or tool is not always restricted to the boundary of the phase in which it is used. To overcome this problem and achieve reliable results from experiments, some factors may have to be measured throughout the development process. This may delay the time scale of the experiment and can make formal experimentation an unattractive option for management.

The Evolutionary Nature of Development Environments

The evolutionary nature of software development environments is evident in the lack of stable and mature procedures of collecting data about the development process. This problem makes Experimentation an even more costly task. However, introducing coherent measurement systems should lead to stabilisation and provide an effective means for monitoring the development process e.g. Kitchenham & Walker [10]. The Japanese Software Factories is a prime example of managing a successful development process using a measurement system which is strictly adhered to, an example in Matsumoto [14].

APPLICATIONS OF FORMAL EXPERIMENTS

It is sometimes unclear to software engineers what areas of software engineering that formal experimentation can be applied to. The following sections identify those areas where formal experimentation can be most valuable.

Confirmation of Theories

There are many "theories" in software engineering that are yet to be experimentally confirmed. These theories are merely supported by anecdotal results or based on "experts" feelings. Some of these theories have penetrated into the industry and have become every-day practice. Formal experiments can be used to confirm or refute these theories, examples in Fenton [6].

Evaluating the effect of methods, tools and techniques

The market is being flooded by new methods and tools each claiming to produce an improvement in quality or an increase in productivity, or both. One way of judging these claims is the use of formal experimentation. This is the main interest of Formal experiments in DESMET.

422 Software Quality Management

Exploration of relationships

The relationships between the different attributes (characteristics) of software systems are of major interest to software developers and project managers. The identification of these relationships is crucial to the success of any quality or productivity improvement programme. Some software attributes such as complexity, modularity and structure are claimed to affect other general attributes such as maintainability. Formal experiments can be used to confirm (or otherwise) a relationship between these attributes and others.

Evaluation of the accuracy of models

Numerous predictive models are being introduced to the software development paradigm. These models are mainly concerned with predicting cost, size and effort. The use of one model sometimes depends on the results of another. For example, predicting the cost of developing a particular software system could depend on the result of another model predicting its size. Formal experiments are one way of exploring the accuracy of these models. This can be achieved by generating actual results against which the predictions of the models can be compared.

Validation of measures

Software measures are normally used to assess the characteristics of some attributes of the development process or the developed software itself. More than one measure can be used to assess the same attribute. The validity of using certain measures to assess a particular attribute can be tested using formal experiments.

A measure is considered to be valid if it reflects the characteristics of an attribute to which it relates to but under different conditions. This can be verified by formalising a hypothesis that states the behaviour of the attribute under these conditions. An experiment can then be conducted to test whether the measure reflects the changes in the attribute when the conditions specified in the hypothesis change.

THE NEW FRAMEWORK

The new framework tries to couple statistical principles of formal experimentation with software engineering terminology in an attempt to provide easy-to-use procedures for conducting experiments. This Framework rests on three main pillars. These are Experimentation Viability, Experimentation Management and Experimentation Procedures.

Experimentation Viability

Although, formal experimentation is an appropriate technique for performing a rigorous evaluation of methods and tools in software engineering, there are situations where it is not viable to employ it. Three main areas govern the

viability of conducting formal experimentation in software engineering: these are Experimentation Requirements, Software Development Requirements and the Cultural Profile of the organisation. A viable situation should not be a compromise position between the three areas, since this will hinder the development of the required system, the reliability of the experimentation results or the commitment of the organisation.

It is when the three requirements intersect that formal experimentation becomes most viable in software engineering. This will lead us to the question of how to identify this intersection. This is the task of the Experimentation Management team. The following sections highlight the inter-relationship between the three areas.

Formal Experimentation Requirements Principles developed in statistics and used in the design of experiments in other disciplines can be employed in formal experiments in software engineering. Three basic principles in experimental design are replication, randomisation and blocking Montgomery [15].

Replication is the repetition of the basic experiment to obtain a more accurate estimate of the treatment's effect.

Randomisation requires that the allocation of experimental materials are randomly allocated and so is the order in which individual trials of the experiment are conducted.

Blocking is a technique used to increase the precision of the experiment. A block is a portion of the experimental material that is more homogeneous than the entire set of material. Blocking helps in making comparisons among the different conditions (state variables) in the experiment.

Applying these principles may involve extra cost to the development budget and/or time to the scheduled plan. Project managers may be reluctant to take on a formal experimentation exercise unless they are compensated in time and cost.

The idea of using Formal Experimentation (as opposed to Case Studies or Surveys) is the high degree of confidence that can be obtained in their results. These results are ultimately used to improve the process or the product in terms of quality or productivity.

Software Development Requirements The development requirements of some software systems can restrict the deployment of formal experimentation as an evaluation technique. For example, if the system to be developed is relatively novel to the developers, then the experimentation results may get diluted by the effect of the learning process. This is different from evaluating the effect of the

424 Software Quality Management

methods/tools on novel developers. Another example is the situation where prototyping is used to develop a system; required modifications to the prototype are usually unpredictable and performed within a tight time scale. This makes the process of simulating a homogeneous environment for the experimental replicates a difficult task.

Despite the highlighted restrictions in the development process, there are situations where the development requirements offer natural grounds for formal experimentation. The development of diverse software systems Bishop, et al, [3] implies that more than one version will be developed for the same problem. This provides a good opportunity for conducting a formal experiment as the cost of replication will be part of the normal development effort.

Some software systems require a high degree of control over their development process. This is sometimes due to the critical nature of their application area (i.e. safety critical systems). The required reliability of these types of systems is of a prime concern to the user, so the development process has to be tightly monitored. The effect of new software methods/tools on the reliability of these systems is very crucial. This effect is best evaluated by formal experimentation.

Cultural and Organisational Profile Ideally, the development process should be stable repeatable and its procedures should be clearly identified to every one in the organisation. Organisations usually strive to achieve this kind of process maturity by introducing measurement systems that monitor the development process Basili & Rombach [1]. Measurement systems should assist in revealing the strengths and weaknesses of the process Rombach & Ulery [18]. Management should ensure that corrective action is taken to overcome weaknesses.

Organisational cultures that encourage the revelation of weaknesses and support corrective actions are bound to speedily arrive at a more mature process. It is in this kind of culture that formal experimentation can be smoothly introduced.

However, in cultures where the revelation of weaknesses are associated with fear and punishment corrective actions can be aimless and ineffective. In this type of culture, formal experimentation is likely to be resisted and driven to failure.

Experimentation Management

It is advisable to appoint a team to overlook all technical, organisational and administrative aspects of the experimentation process. The main tasks of the team are:

- to identify the viability of formal experimentation,
- to design the experiment; and
- to manage the implementation of the experiment.

There are four objectives to be achieved by forming a management team. These are to:

- Guarantee early involvement by the users (developers);
- Ensure that formal experimentation principles are adhered to;
- Secure management commitment ; and
- Take the administrative task of the experiment off the developers.

The team members should be representatives of the interests of the experimentation requirements, the development requirements and the cultural and organisational profile. The team size should be between two and five with an optimum number of three. The team members could be described in terms of their roles as follows.

The beneficiary This person represents the group of people who will directly benefit from the findings of the experiment (i.e. the developers). S/he should have reasonable experience on the "ground". The involvement of the beneficiary at the beginning of the experiment ensures that the real problems facing the developers in the development process are not ignored and are accurately reported. Another major benefit is that the ideas of the users on how to improve quality or increase productivity are taken on board. This first hand experience is most valuable in diagnosing and characterising the problem to be evaluated.

The evaluator This is an individual who is explicitly charged with the responsibility of conducting the experiment and coming to a conclusion. This person will be expected to have some knowledge of formal experimentation principles and requirements. However, the proposed framework will provide the evaluator with all the basic procedures needed to conduct a formal experiment in a step-by-step fashion.

The Champion It is important that the treatment should have a champion. This a member of the organisation, preferably an influential one, who is committed to the treatment. While, for the purpose of the experiment it may be necessary to regard the treatment initially in a rather speculative light, it is equally important that the champion should have an optimistic view of its potential benefits. It is also important that the champion has a very clear idea of the cultural and the organisational profile of the institution in which the experiment is to be undertaken. It is worth going to considerable efforts to secure the commitment of management to the experimentation process.

426 Software Quality Management

Experimentation Procedures

The main aim of this section is to provide a non-technical approach to conducting formal experimentation in software engineering environments with easy to use procedures.

A major benefit of formal experiments is the high degree of confidence that can be obtained in their conclusions. However, there are difficulties in simulating an experimental environment that resembles a real life situation. The framework provides a step-by-step approach to assist the experimenter in overcoming this problem.

Seven rules for conducting an experiment The procedures are presented in a step-by-step fashion with an allowance for feedback. The success of the implementation of these procedures depends on satisfying Seven "Golden" Rules. These are:

- One Keep the defined purpose of the experiment always as the main term of reference
- Two Use Direct measures whenever possible
- Three Plan the experiment in consultation with the practitioners.
- Four Use maximum automation possible to collect data and record results
- Five Make use of baseline data if they exist
- Six Trial the experimental task in a "dummy run" before using it in earnest.
- Seven Ensure that all participants are fully briefed in good time and kept informed throughout the experiment

The Experimentation Plan To conduct formal experiments, it is important that the experimenter has a clear plan that guides him/her throughout the experiment. This plan should be drawn at an early stage of the experiment. The following procedures are recommended to obtain such a plan.

Step One: Purpose Definition Clear statements of the objectives of experiments substantially enhance the experimental design process and later the understanding of the results. Although this is a rather common sense point, it is often omitted or considered at a late stage when it is very costly to modify the design to suit the objectives.

A clear statement of the objectives should be considered before embarking on any type of experimental technique. The importance of purpose definition has led to the development of an paradigm for measurement based on goals (Goal/Question/Metrics) e.g. Basili & Weiss [2] and Basili & Rombach [1].

Step two: Focus the Scope It is sometimes tempting to widen the scope of experiments in order to get more results. The danger is that widening the scope may dilute the results. It is advisable to have a more focused scope for the initial experiment. Further experiments could be conducted to explore interesting aspects that the initial experiment may reveal. Also, widening the scope of the experiment may result in a higher number of replications and a longer duration. Inevitably, this will lead to an increase in cost.

In DESMET, the scope of an experiment could be characterised using a number of factors such as the development environment, phase of the development, type of application and type of treatment. These factors will be crucial for the identification of state variables. The particular aspects of quality or productivity under consideration should also be defined at this stage as they will serve as basis for the identification of response variables.

Step three: State the Hypothesis A hypothesis is a statement which explicitly proposes that the application or the manipulation of a particular treatment causes changes to some attributes, called response variables, that are of interest to the experimenter. For example, structured programming (the treatment) improves maintainability (the response variable).

Step four: Define The Treatment This is the method, tool, or technique to be evaluated. If more than one treatment is to be evaluated, then all treatments have to be explicitly specified. The treatments should be appropriate to the evaluation environment. It is necessary to check that all requirements for the application of the treatment are available (e.g. training, equipment, personnel ... etc.).

Step five: Identify Response Variables These are the attributes of the product or process that will be affected by the treatment. One attribute can be characterised by measuring more than one variable. For example, maintainability could be characterised by measuring effort spent on changing a module to meet new requirements and by cost of correcting an error. It should be decided at the start of the experiment which measurements will be used to assess the response variables. These measurements must be consistently used throughout the experiment.

Step six: Select State Variables These are other variables that may influence the application of the treatment and therefore the results of the

428 Software Quality Management

experiment. These variables are not directly related to treatment. They are mainly related to the developer, the product and the process.

The developers represent the subjects of the experiment. State variables related to subjects include years of programming experience and training on the use of the treatment.

The product represents the experimental objects (the software to be developed or the development material to be exposed to the treatment). State variables related to the product include program size and type of application.

The process represents the experimental task. State variables related to the process could be team size, project size, subjects' experience and computer support. These variables must be controlled or factored out. This can be achieved by using the principles of randomisation and blocking.

Step seven: Define Data Collection and Measurement Procedures Consistent procedures for data collection must be used throughout the experiment. It is also important to specify the measurements and metrics to measure each variable. It is advisable to automate the data collection effort whenever possible. Also, the choice of measurement should be carefully considered as only those measures directly related to the variable should be employed. Deploying large number of measurements, hoping that one of them may pick up an interesting feature, could have a negative effect as developers may be diverted from the initial evaluation objective by an overwhelming task of data collection. Further details appear in the DESMET Handbook of Data Collection and Metrication Kitchenham [8].

Step eight: Choosing Experimental Objects These are the objects that will be exposed to or manipulated by the treatment. For example, if the treatment to be evaluated is a testing method then the experimental objects will be the programs that will be tested using the treatment.

The evaluator needs to identify all the required experimental objects and their availability. All experimental materials should be homogeneous (similar or the same) to avoid discrepancies in the results which could be mainly due to differences in the experimental materials. They could be measured or characterised (e.g. by size or effort) to ensure their suitability.

Step nine: Choose Experimental Subjects These are the developers who will perform the experimental task. To avoid differences in ability, they can be randomly chosen from a bigger population and/or randomly assigned to the experimental group (teams).

CONCLUSION

This paper proposes a framework for applying formal experimentation in the evaluation of software methods and tools. The framework deals with the establishment of the viability of such experimentation together with the management and conduct of experiments. Within the framework an attempt is made to recognise and overcome some of the practical problems which can arise when trying to initiate experiments within a software development environment.

The framework does not stand on its own but depends on other modules under development within the DESMET research project. In particular technical support is required in the area of data collection and analysis. Future work will concentrate on validating the framework via field trials and on integrating this within a more general evaluation framework.

6. References

1. Basili, V.R., and Rombach, H.D. 'The TAME Project: Towards Improvement-Oriented Software Environments', *IEEE Transaction on Software Engineering*, Vol. 14, No 6, 1988.
2. Basili, V.R., and Weiss, D.M. 'A Methodology for Collecting Valid Software Engineering Data', *IEEE Transactions on Software Engineering*, vol. SE-10, No. 6, pp. 728-738 November 1984.
3. Bishop P.G., Esp D.G., Barnes M., Humpherys P., Dahll G., and Lahti J., 'PODS- A Project on Diverse Software', *IEEE Trans. on Software Engineering*, vol. SE-12, 9, 1986.
4. Card, D.N., McGarry, F.M. and Page, G.T. 'Evaluating Software Engineering Technologies', *IEEE Transactions on Software Engineering*, Vol.SE-13, No.7, pp. 845-851, July 1987.
5. DESMET, Determining an Evaluation Methodology for Software MMethods and Tools, State of The Art Report, DTI, 1991.
6. Fenton, N. *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, 1991.
7. Gilmore, D.J., 'Methodical Issues in the Study of Programming', *Psychology of Programming*, Editors J-Michael Hoc, T.R.G. Green, Samurcay, D.J. Gilmore, Academic Press, 1990
8. Kitchenham, B.A, DESMET Handbook of Data Collection and Metrication, Book 1: Software Measurement Goals, internal project report, 1992.



430 Software Quality Management

9. Kitchenham B.A, Kitchenham A.P., and Fellows J.B, 'The Effects of Inspections on Software Quality and Productivity', ICL Technical Journal, May 1986.
10. Kitchenham B.A. and Walker, J.G., 'A quantitative Approach to Monitoring Software Development', Software Engineering Journal, January, 1989.
11. Law, D., and Naeem, T., 'DESMET: Determining and Evaluation methodology for Software Methods and Tools', *Proceedings of BCS Conference on CASE - Current Practice, Future Prospects*, Cambridge, England, March 1992.
12. Merlet, J.M., and Dupont, C., 'Measurement of Quality on Clerical Systems: A Case Study', pp. 341 to 349, *Proceedings Eurometrics'91 Conference*, Paris Mar. 1991.
13. Mohamed W. A., and Sadler, C.J., 'Methodology Evaluation: A Critical Survey', pp. 101 to 112, *Proceedings of Eurometrics'92 Conference on Quantitative Evaluation of Software & Systems*, Brussels, April 1992.
- 14.. Matsumoto Y., 'An Overview of Japanese Software Factories', in Japanese Perspectives in Software Engineering, (Ed. Matsumoto, Y. and Ohno, Y.), Addison-Wesley, 1989.
15. Montgomery, D. C., 'Design and Analysis of Experiments', 2nd edition, JohnWiley & Sons, Inc., 1984.
16. Panzl, D.J., 'A Method for Evaluating Software Development Techniques', The Journal of Systems and Software, 2, 1981.
17. QUANTUM, A measurement-based framework for the assurance of software quality, DTI, 1992.
18. Rombach, H.D., and Ulery, T.B., 'Improving Software Maintenance Through Measurements', Proceeding of The IEEE, Vol. 77, No.4, April 1989.
19. Schach, S.R. 'Methodology Characteristic Frameworks and Software Specification and Design: A Critique of "Methodman II"', pp. 196-200, *Proceedings of the Fourth International Workshop on Software Specification and Design*, IEEE, 1987.
20. Selby, R.W., Basili, V.R., and Baker, F.T., 'Cleanroom Software Development: An Empirical Evaluation', *IEEE Trans. on Software Engineering*, vol SE-13, 9, 1987.