Georgiadou Elli, Siakas Kerstin V., Berki Eleni (2003). Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development, *EuroSPI 2003* (European Software Process Improvement Conference), Graz, Austria, 10-12.12.2003, pp.IX 31-45

Quality Improvement through the Identification of Controllable and Uncontrollable Factors in Software Development

Elli Georgiadou[1], Kerstin Siakas[2], Eleni Berki[3]

[1] Middlesex University, School of Computing Science, Trent Park, London, N14 4YZ, UK (e.georgiadou@mdx.ac.uk)

[2] Technological Educational Institution of Thessaloniki, Department of Informatics, P.O. Box 14561, GR-54101 Thessaloniki, Greece (siaka@it.teithe.gr)

[3] Department of Computer Science and Information Systems, P.O.Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland (eleberk@cc.jyu.fi)

Abstract

The software engineering community has moved from corrective methods to preventive methods shifting the emphasis from product quality improvement to process quality improvement. Inspections at the end of the production line have been replaced by design walkthroughs and built-in quality assurance techniques throughout the development lifecycle. Process models such as the Spiral, V, W and X-Models provide the principles and techniques for process improvement which, in turn, produces product improvement.

Factors that affect the quality of software need to be identified and controlled to ensure predictable and measurable software. In this paper we identify controllable and uncontrollable factors and provide empirical results from a large industrial survey, as well as conclusions relating to the models and metamodels for the estimation, measurement and control of the totality of features and characteristics of software.

Keywords

Metamodelling, process improvement, controllable factors, uncontrollable factors, software quality, development methods, soft & formal methods, software metrics, culture

1 Introduction

Software is at the heart of most modern businesses. Business success depends on the quality, the cost and the timeliness of the software they use. A Software Quality Management System is the enabling mechanism within an organisation which co-ordinates and controls the functions needed to achieve the required quality of product or service as economically as possible. It will involve every organisational function that directly or indirectly affects a delivered product or service [DTI, 1992]. For a successful implementation of a Software Quality Management system, weak areas in the current situation have to be identified and gradually improved by introducing a formal process suitable for the specific organisation.

A Software Quality Management system requires that software engineering knowledge and discipline will be applied at all phases of the development life cycle in order to assure software quality. In Software Quality Assurance (SQA) software metrics are used to help numerically determine the quality of both the process and the products together with an independent verification and validation (V&V) usually carried out by the software quality assurance engineers. The right measures visualise different problems in the organisations and after taking action they are a means of control. Verification is the process of determining if the products of a given phase of the software development cycle fulfil the requirements for the next phase regarding consistency, correctness and completeness. Validation is the process of evaluating software throughout its development process to ensure compliance with software requirements. Thus software quality assurance engineers are required to possess sufficient domain knowledge so that they can evaluate the completeness and correctness of system requirements. They must also have the ability to determine whether the design has incorporated all requirements accurately. Finally they are responsible for advising management when or whether a product is reliable and meets quality standards.

There are essentially two approaches that can be followed to ensure product quality, one being quality assurance of the process by which a product is developed (ISO 9000:2000), and the other being the evaluation of the quality of the end product (ISO 9126). Both approaches are important and both require the presence of a Software Quality Management system. The choice of Software Quality Management system, processes, standards and metrics greatly depends on the organisational culture and the maturity level of the organisation [Siakas, 2002].

During the last two decades, there has been a change from concentrating on the product to concentrating on the development process. Software Process Improvement (SPI) has become a practical tool for companies where the quality of the software is of high value [Järvinen, 1994]. In a Technical report Herbsleb et al. [1994] showed (with results from 13 organisations) that due to SPI the products (the number of post-release defect reports were used as a measure) and business value (ROI, Return On Investment) was improved. It is generally considered that, a well-documented and a repeatable process is essential for developing software products of high quality. There is also evidence that the use of standards and process assessment models has a positive impact on the quality of the final software [Kitchenham and Pfleeger, 1996] Evidence for the emphasis on process is also hat ISO certification does not certify product quality but that a stated and documented process is followed.. Similarly, well-known and recognised assessment models like CMM [Paulk et al., 1993; Paulk, 1993; 1995], BOOTSTRAP [Haase, 1992; Haase and Messnarz, 1994; Kuvaja et al.,1994].

2 The Selection of a Suitable System Development Method

Among the most important criteria for selecting a method is its 'suitability for modelling the application domain' and the 'ability to transform its characteristics in a consistent and correct manner to a computable design/software architecture' [Jackson & Zave, 1993]. Different systems need different techniques to *capture* their properties and similar or different languages to *model* and *preserve* them. Most systems' developers either use methods (with or without automated tools), which are technically oriented (hard methods) or try to concentrate mainly on human factors by using human activity oriented methods (soft methods).

According to Holcombe and Ipate, there is *"little empirical evidence of the superiority of one method over another in large-scale projects"* and moreover there *is "a crisis of intellectual respectability in the subject"*. They continue supporting that *"not only the evaluation of the methods used is weak, the selection of the types of system and problem to focus on is very restrictive. In order to convince, in a scientific manner, that method A is better than method B in large design projects (and that is where the problems are), we must present rigorous evidence drawn from carefully controlled experiments of suitable complexity. This is, more or less, impossible in practical terms. Is there an alternative approach? The use of theoretical models of computing systems can provide some alternative approaches ..."* [Holcombe & Ipate, 1998].

2.1 Limitations of IS development methods and associated metamodels

There are two important problems associated with IS methods currently available. Firstly, the absence of the *interconnections between hard and soft problems,* which inevitably leads to inadequate information systems with problematic functionality. Secondly, the design or software architecture of a system lacks those *computational characteristics* because they are not specified as an integral part of it. These characteristics make it reliable, easily re-engineered and maintained. We might ask whether there is any all-stakeholder-oriented method among the plethora of methods to adequately capture all the above requirements. And if so, is there any adequate automated support through an Integrated Computer Assisted Software Engineering (I-CASE) environment during all the phases of the software engineering lifecycle?

Development methodologies in order to be reliable, need to also have solid foundations in their generic structure for testing and mapping of the requirements from design to implementation in an expressible, correct and consistent way. They should also cater for systems and method re-engineering with principles that accommodate change regarding soft and hard components of the system such as people's opinions, systems' processes, and so on. One of the most important properties to establish quality assurance of the process models that handle the previous is the ability to provide *formal testing* which is also *absent* from existing frameworks and *metamodels* [Berki, 2001]

Moreover, considering the principles and different levels of metamodelling abstraction, that Method Engineering as a discipline caters for, none of the metamodelling environment and notations handles metadata *and* metaprocess specification aspects for method modelling considering methods as dynamic models. This is an invaluable property allowing modellers to keep control on both process metamodelling and metadata of a method because they can express it at the same time. Arguably, when we built a method, we need a general and generic process model of abstraction such as to offer us both generality and specialisation for different application domains [Berki *et al.*, 2001].

Method Engineering is of course a new division/discipline in software development and its emergence was clear while there was a need to improve process models (methods) for IS development. However, the efforts were concentrated mostly on issues such as integration of methods, flexibility of method construction, new theories on requirements management and so on, whilst emphasis was never placed on the relation of recent disasters of software development such as 'the millennium bug', or the space rockets and railway disasters.

The previous are examples of large software systems infrastructure that we have inherited and there is an urgent need to re-engineer, test and improve them. These and similar incidents were due to software design errors and their study and examination revealed that there is urgent need for these to be addressed at method engineering level and from the metamodelling (and possibly metametamodelling) point of view.

In the field of method metamodelling most methods' metamodels are led by rules of modelling abstraction which concentrate on soft or hard issues for the evaluation of methods, quantitatively or qualitatively. However, many of implied goals and objectives are not addressed and remain in the wish lists, whilst it seems that many process-oriented problems in software development remain undefined or not specified at all, to say the least. Some others such as the problem of testing remain in silence.

"*There is no single metamodelling initiative or CASE-Shell tool, which claims to facilitate testing"* [Berki, 2001].

In general, the use of formal methods for developing software-based systems did not lead to quality information systems, in the past. It soon became clear that analysing and designing a system with formal methods offers some quality assurance regarding the development of unambiguous, consistent, correct and verified mathematically-proven specifications, but there were other issues raised. The most frequently mentioned problem that is associated with the use of most formal methods in software development is the unfriendly and fragmented approach, which prevents wide understanding and results in high costs for training and prototype construction and testing.

Huotari & Kaipala summarised results of scientific work within the field of Human-Computer Interaction (HCI) focusing on cognitive aspects on methods' use. Their review analyses and synthesises the main contributions and takes a critical view on how cognitive aspects are considered and what methods are used. They believe that *"Despite a trend of applying cognitive task analysis and other user-centered system design methods, issues of human cognition and human information processing still need more attention in the IS research."* [Huotari & Kaipala, 1999].

2.2 The Need for a Multidisciplinary Approach in IS Development

Exposing the issues that were examined in the previous paragraphs and having research evidence on the cultural factors that influence software development (Ref. Kerstin's PhD) we need to consider the following: The *capturing of the needs' interconnected nature in a holistic question*, which will connect them to the needs of IS. Such a question will draw examples and provide links, opinions and insights from various related contexts and contents, in order to finally present an integrated solution [Berki *et al.*, 2003a].

That being the reality, a rigorous and integrated method based on holistic communication rules must also provide appropriate syntactic and diagrammatic structures to model the semantics, pragmatics and semiotics of systems' and stakeholders' cultural requirements and thus provide the scientific ground for *usability engineering* [Berki *et al.,* 2003b].

Therefore, the interest for IS designers should be in identifying and using general and understandable, groupware-oriented structures that capture adequately the features of specification and computation. This ca be achieved in terms of specialised and sufficiently general design structures that can capture the richness and testedness of domain specifications, considering at the same time people's cognitive needs for maximum participation and, therefore, *empowerment*.

It is important for the software developer to state clearly their objectives for software product or the process improvement, and to specify the product/process response characteristics that reflect these objectives. The formulation of the problem as well as the production of a list of controlled parameters and noise variables can be achieved through brainstorming and formulated using techniques such as the Ishikawa (cause and effect) or fishbone diagram.

Measurable objectives should be chosen such as the number of bugs found during formal inspections which are conducted during the software life cycle under the specified methodology a company adopts.

2.3 Sensitisation towards human factors

Throughout its lifetime, Information Systems Engineering has failed to deliver software products at satisfactory levels of reliability, timeliness and cost. To solve this problem, the Information Systems industry has turned its attention to the software development process, arguing, like other industries before it, that a high-quality process will deliver high-quality products. Thus various Software Quality

Management Systems have been proposed and developed. Although all these involved technological elements (which have generally received much attention), most people now recognise that the successful development of a quality culture depends more heavily on social factors. Yet these social factors have been largely neglected. However, in recent years a shift has taken place in software development by taking more human factors into consideration. Cornford and Smithson [1996] for example argue that we must see beyond any technology if we are ever to understand what happens when Information Systems are built or operate. They emphasise that people, people-structures and people-processes have to be taken into consideration when adopting an Information System's perspective both in development and research. A similar approach is emphasised by Klein and Lyytinen [1985], who state that one of the reasons for an Information Systems' failure is that the design conflicts with prevailing organisational culture and attitudes.

One of the human factors is commitment, which has long been argued to play a major role in the success of software projects since it increases the odds that appropriate actions will be taken [Ginzberg, 1981; Kautz, 1999; Lucas 1981; Marcus 1981]. Most of the researchers in the field seem to agree that commitment is one of the most important human factors in determining whether a well-planned process improvement program will succeed or fail [Dahlberg and Järvinen, 1997; Diaz and Sligo, 1997; Grady, 1997; Humphrey, 1995; Rodenbach et al., 2000; Stelzer and Mellis, 1998; Zahran, 1998].

Another human factor is culture. The main objective of the study described above [Siakas, 2002] was the identification of a number of cultural factors that have a bearing on the successful adoption and implementation of a Software Quality Management system. The main contribution of the study was the development of the SQM-CODE (Software Quality Management: Cultural and Organisational Diversity Evaluation) model, which assesses the fit between national and organisational culture.

3 *Case Study*

A comparative study was carried out in the form of quantitative and qualitative investigation in four counties, namely Denmark, Finland, Greece and the UK. The quantitative investigation was a survey collecting hard data by using a postal questionnaire. In parallel, a qualitative method in form of case studies was performed in order to address different aspects of the research problem, to confirm the findings from the questionnaire and to prove the hypothesis.

The questionnaire was sent to organisations developing software for own use or for sale. Totally 307 questionnaires were completed. In addition field-studies were undertaken in several organisations. Totally 87 interviews were conducted in Finland, Denmark and Greece with software developers at different levels and with different positions in the organisations. Following the initial verification phase observations were carried out in a Danish organisation for a period of two month [Siakas and Balstrup, 2000]. The objective of using observations was to investigate in more depth the research problem and to verify the findings.

3.1 Are measures of the quality of software process kept?

From figure 1 we observe that amongst the organisations taking part in the study Greece is the country that keeps measures of the quality of the software development process to highest degree. The sum of the values for quite a lot or very much so is 61.9% for Greece, 44.7% for Denmark, 42.6% for Finland and 42.5% for the UK.

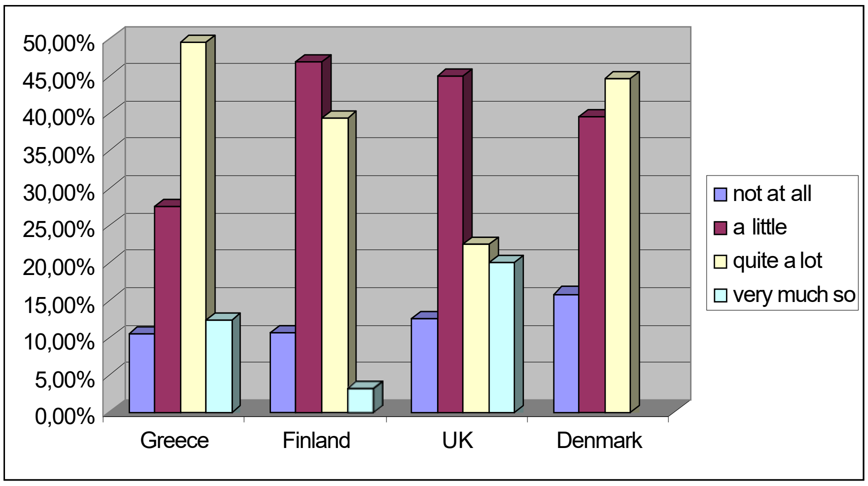

Figure 1: Measures of the quality of the software process

The significance of the Chi-square is 0.002, which indicates that the null-hypothesis, that the responses are similar for all countries, can be rejected. This means that we have statistically proved that there are significant differences in responses depending on country of origin.

3.2 Are measures of the software product kept?

From figure 2 we observe that amongst the organisations taking part in the study Greece is the country that keeps measures of the quality of the software product to highest degree. The sum of the values for quite a lot or very much so is 67.3% for Greece, 50.5% for Finland, 50% for Denmark and 47.7% for the UK.

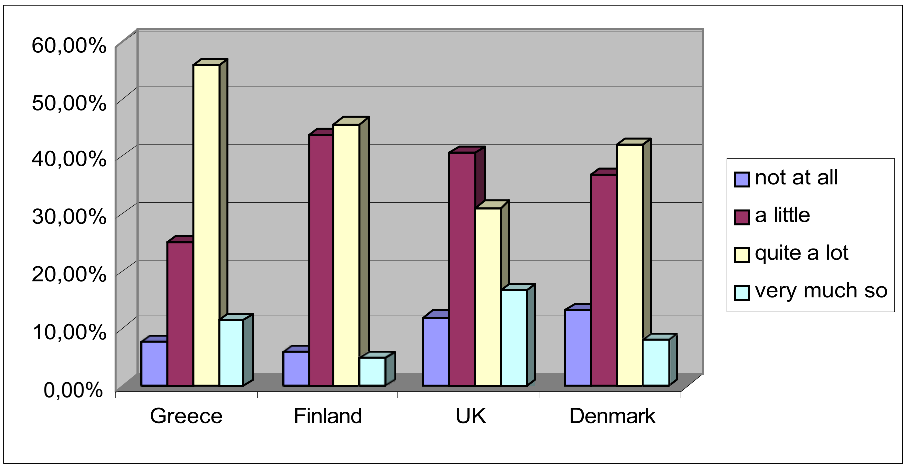

Figure 2: Measures of the quality of the software product

The significance of the Chi-square is 0.042, which indicates that the null-hypothesis, that the responses are similar for all countries, can be rejected. This means that we have statistically proved that there are significant differences in responses depending on country of origin.

In order to understand if there is a relationship between measurement of software process and measurement of software product table 1 is presented.

	Measure of quality of software product					Total
		not at all	a little	quite a lot	very much so	
Measure of quality of software development process	**not at all**	21	11			32
	a little	2	71	29	4	106
	quite a lot		14	93	7	114
	very much so		3	6	15	24
Total		23	99	128	26	276

Table 1: Cross-tabulation of measures of the quality of the software process vs. product

From table 1 we observe that measures the quality of the software process also tend to measure the quality of the software product. There is also a significant correlation on

The Pearson's correlation is also significant at the 0.01, which proves that there is a significant relationship between measurement of the quality of the software development process and the measurement of the quality of the software product.

3.3 What is the degree of impact of international, national or organisation specific standards and systems on measurement?

Figures 3 to 6 present the impact on the degree of measurement by quality systems, like ISO 9001 and TickIT, as well as in-house standards have any use figures 3 to 6 are presented.

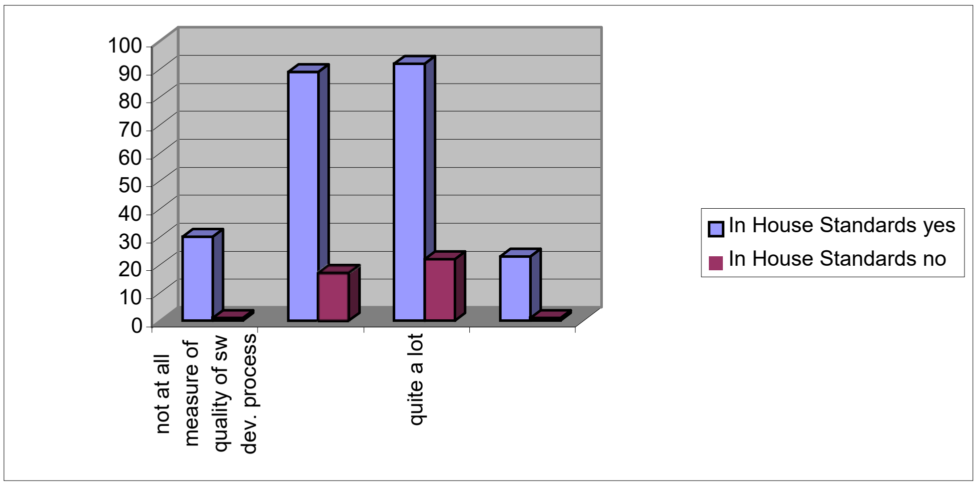

Figure 3: Measurement of quality of software development process vs. in-house standards

From figure 3, which is a cross-tabulation of measurement of the quality of the software development process and the use of in-house standards we observe that the majority of the organisations, which have in-house standards measure the quality of the software development process a little or quite a lot. The surprising element is that 30 (of 275) respondents which responded that they have an in-house quality system also answered they do not measure the quality of the software development process at all. The significance of the Chi-square is 0.061.

From figure 4, which is a cross-tabulation of measurement of software development process and third party assessment we observe that most of the organisations do not have any third assessment at all. However this does not seem to have any greater impact on if they measure the quality of the software development process or not. The significance of the Chi-square is 0.180.

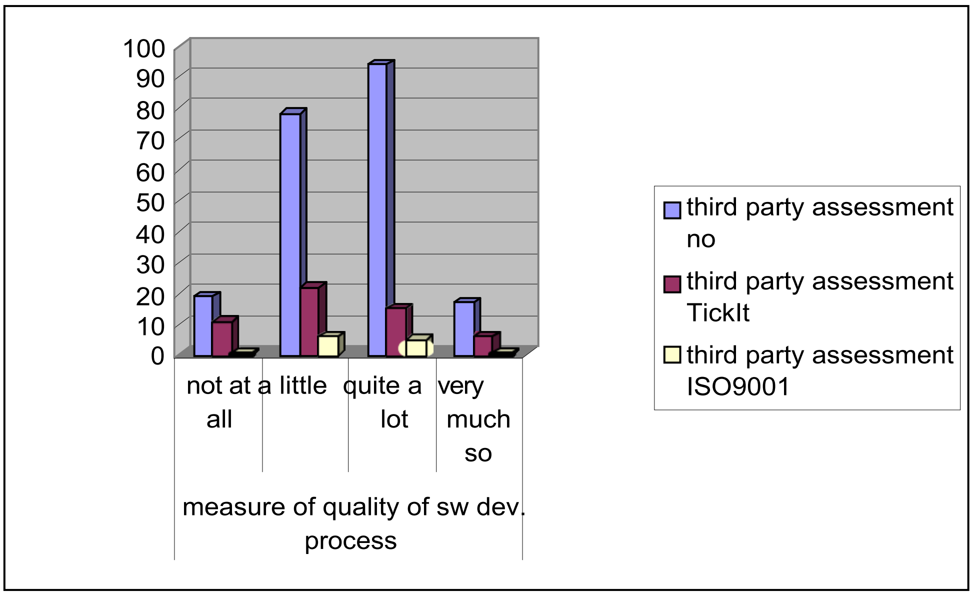

Figure 4: Measurement of quality of software development process vs. third party assessment

From figure 5, which is a cross-tabulation of measurement of the software product and in-house standards we observe similar results as with figure 3. The significance of the Chi-square is 0.012

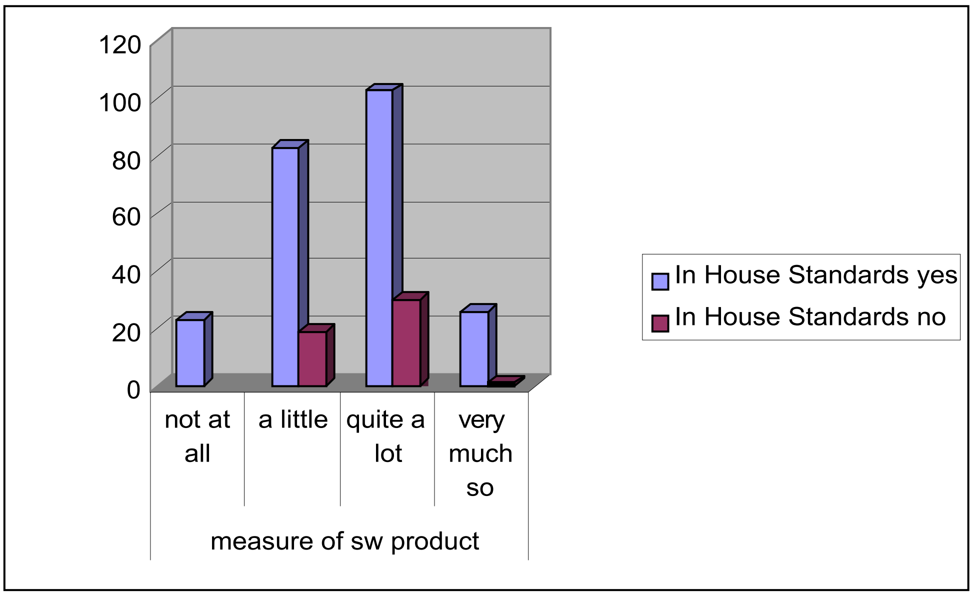

Figure 5: Measurement of quality of software product vs. In-house standards

From figure 6, which is a cross-tabulation of measurement of software product we observe similar results with figure 4. The significance of the Chi-square is 0.254.

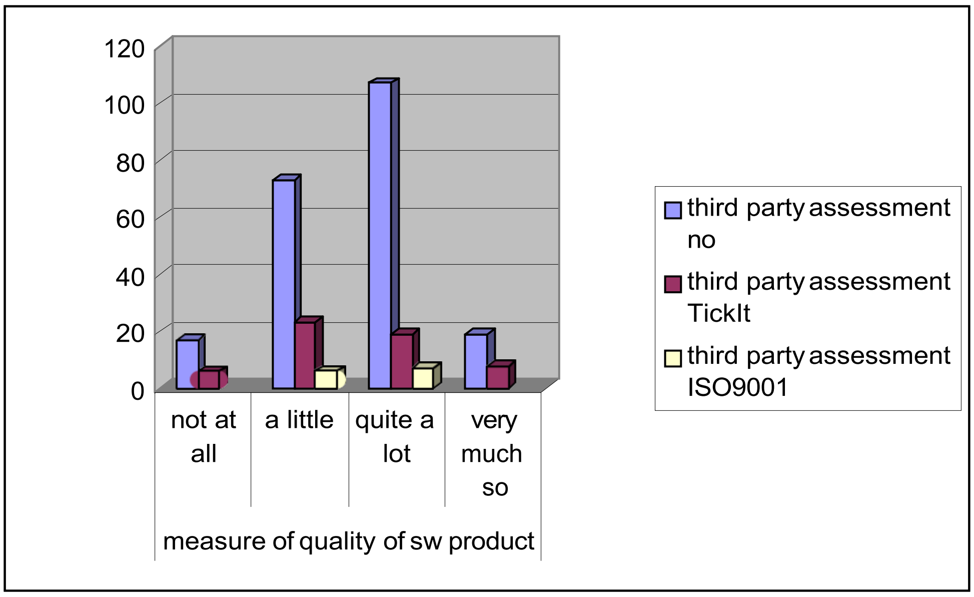

Figure 6: Measurement of quality of software product vs. Third party assessment

4 *Software Measurement and Software Metrics*

"To measure is to know. If you can not measure it, you can not improve it."(William Thomson (later Lord Kelvin) (1824 - 1907).

Managing the development process requires the collection of suitable metrics which will provide insights into the strengths and weaknesses of the process. What to measure, how to measure, when to measure are the fundamental questions which need to be addressed.

According to Kitchenham [1996] “Software Metrics can deliver :support for process improvement, better project and quality control and improved software estimation”. *Direct measurement* of quality factors is often possible very late in the life cycle. For example *reliability*, which is concerned with how well as software system functions and meets a user’s requirements, can be measured after that the software has been used for a stated period of time under stated conditions, while *indirect measurement* of quality, like number of discrepancy reports (deviations from requirements) can be obtained earlier in the life cycle. Other estimates of quality can be made by developers even earlier than the indirect measurements of quality.

According to ISO-9126 software quality may be evaluated by six characteristics, namely functionality, reliability, efficiency, usability, maintainability and portability. Each of these characteristics is defined as a “set of attributes that bear on” the relevant aspect of software and can be refined through multiple levels of sub-characteristics[6]. Definitions of sub-characteristics are given in Annex A (of the standard), which is not a part of the International Standard. Attributes at the second level of refinement are left completely undefined.

In order to achieve improvements in the software process and the software product we need to understand, measure and hence control the variability to the desirable/achievable degree of confidence. Measurement is defined as the process of assigning symbols, usually numbers, to *represent* an *attribute* of the entity of interest, by *rule* [Fenton 1991], [Shepperd 1995]. Entities of interest include objects, (e.g. code, specification, person) or processes (e.g. analysis, error identification, testing). Distinct attributes might be length of code, duration, costs. Representation is usually in numbers (or other mathematical objects e.g. vectors). Finally in order to provide objectivity we need to assign numbers (symbols) according to explicit rules: how to choose which symbol should represent the attribute. Such rules ensure that the assignment is not random.

Fenton and Pfleeger [Fenton 1997] provide a refined definition of measurement: “Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterise them according to clearly defined rules. The numeral assignment is called the measure.” Hence, in order to understand the definition of measurement in the software context, we need to identify the relevant entities and attributes which we are interested in characterising numerically.

4.1 Controllable Factors

In order to understand and control the process we need measurements of both the current and the desired/new system. Internal metrics [Fenton, 1991] an be obtained in terms of the product (code) and they are counts (such as LOC, NO of Classes, McCabe Complexity) and ratios (such as No of calls Module, Average length of hierarchy). Additionally, these metrics can be generated automatically by using tools such as CANTATA, Testbed and Logiscope.

Attributes such as the morphology, architectural structure, depth of class hierarchy, size of module, maximum level of module complexity etc. can be contolled through a management mechanism and specific guidelines to the developers. *Controllable* design parameters can be found in the software development process, the software product and the software development environment [Kitchenham, Fenton, Barbor & Georgiadou].

However, external attributes [Fenton 1991], [Kitchenham, 1996], [Georgiadou 1999, 2001] which are behavioural such as understandability, maintainability are more elusive and more difficult to measure. Metrics for these attributes are both qualitative and quantitative. They are almost always obtained indirectly through the use of surrogate measures [Kitchenham, 1996], [Georgiadou 1993, 2001]. For example maintainability can be estimated, calculated and controlled through measuring the time taken for a specified maintenance task. Results obtained by Georgiadou et.al in a series of controlled experiments provided confidence (through statistical methods) in our ability to effectively use surrogate metrics [Georgiadou, 93, 94, 97, 98, 2001].

4.2 Uncontrollable Factors

Human factors are unpredictable and mostly difficult, often impossible to control. For example, performance variability in a human being, such as his/her experience and communication skills needed in a software development team. The developers’ performance has an effect on producing quality software products in a similar way to the effect of machines on the manufacturing of products. It is important to maximize and properly maintain programmers’ performance. The possible control factors will be conducting educational sessions within and outside a company where software developers are encouraged to learn the new techniques of their interest or polish their skills. Recreational events may help developers to get to know each other better and this will be reflected in better communication and teamwork in an office. At the extreme, the design of the office environment itself is investigated. Changing the type of chairs in current use to the ones designed to ease backpain caused by sitting all day will be welcomed by the developers (ergonomics). The temperature and humidity in the workplace also can affect the developers’ performance. Therefore the suggestions made here must be investigated in the software industry.

Experimental evaluations carried out by Basili [1986]), Shepperd [1995], Georgiadou [1999, 2001] attempted to identify design parameters and hence factors, which can be controlled. According to Taguchi in [Logothetis 1989] it is desirable to choose the set of design parameters, which are less affected by the variability of these factors. For example, developers’ experience can be controlled to certain extent by years in profession and looking at the past projects involved. However every individual is unique. His/her capability, patterns of learning and cognition are likely to be different from those of others of similar experience. The health of the developers may effect on their performance at work.

5 Summary and Future Research and Development

The fundamental philosophy of this research work is that it addressed the needs for adequate expression of process models within various cultural factors and different organisational communication. In doing so, we interconnected and commented on their dynamic and computational characteristics connecting them to the coverage of testing and re-engineering.

Our multidisciplinary study reveals the strengths that the *holistic* nature of such an approach provides software development with the use of software measurement as the instrument for understanding, estimating and controlling the quality of specified factors. Bearing in mind that different stakeholders place different emphasis on software attributes we provide flexible and hence customisable quality requirements.

The efficiency of a software product such as execution time has high priority as a software quality factor. To maximize the performance of a product, the choice of machines, operating systems and programming language has to be included in the list of parameters.

Enhanced Reliability (usually achieved through testing, walkthroughs, reviews and inspections) will reduce productivity and will therefore increase costs. Both of these cause losses to the sponsor. Enhanced functionality increases costs (in the short term) and causes losses to the sponsor.

Usability is enhanced through greater understandability, which in turn is enhanced through design correctness and consistency and through training, on-line help and support all of which reduce productivity with the exception of CBD which makes extensive reuse of code and increasingly reuse of designs too.

It is important for the software developer to state clearly their objectives for software product or the process improvement, and to specify the product/process response characteristics that reflect these objectives. The formulation of the problem as well as the production of a list of controlled parameters and noise variables can be achieved through brainstorming and formulated using techniques such as the Ishikawa (cause and effect) or fishbone diagram.

Measurable and hence controllable objectives should be chosen such as the number of bugs found during formal inspections, which are conducted during the software life cycle under the specified methodology a company adopts.

Future investigations will concentrate on the development of meta-CASE tools for the specification, implementation, re-engineering and metrication of both the process and the product of software development across problem domains, national, cultural and technical boundaries.

6 Literature

Barbor, N & Georgiadou, E. [2002] Investigating the applicability of the Taguchi Method to Software Development, Proceedings of Quality Week, San Francisco. USA, July 2002

Basili, V., R. W. Selby, & D. H. Hutchins[1986] Experimentation in Software Engineering. IEEE Trans. on Software Engineering, SE-12. p. 733-743.

Berki, E., Georgiadou, E., Holcombe, M.[2003a]: "Process Metamodelling and Method Engineering as Tools for Improved Software Quality Management - *A Chronological Review and Evaluation Critique Considering the Need for a New Scientific Discipline*",In the Proc. of Ross, M., Staples, G. (Eds.) 11th International Conference on Software Quality Management, SQM 2003, Glasgow, April 2003.

Berki, E., Isomäki, H., Jäkälä, M.:[2003b] "Holistic Communication Modelling: Enhancing Human-Centred Design through Empowerment", in the Proc. of the HCI International Conference, University of Crete, Greece, June 2003

Berki, E., Lyytinen, K., Georgiadou, E., Holcombe, M., Yip, J.:[2002] "Testing, Evolution and Implementation Issues in MetaCASE and Computer Assisted Method Engineering (CAME) Environments", In the Proc. of King, G., Ross, M., Staples, G. & Twomey, T. (Eds.) Issues of Quality Management and Process Improvement, 10th International Conference on Software Quality Management, SQM 2002, Limerick, Ireland, March 2002.

Berki, E.: [2001]"Establishing a Scientific Discipline for Capturing the Entropy of Systems Process Models, CDM-FILTERS: A *Computational* and *Dynamic Metamodel* as a *Flexible* and *Integrated Language* for the *Testing, Expression* and *Re-engineering* of *Systems",* Ph.D. Thesis, Faculty of Science, Computing & Engineering, University of North London, 2001.

Burr A., Georgiadou E.[1995] *"Software development maturity - a comparison with other industries",* 5th. World Congress on Total Quality, India, New Delhi, Feb. 1995

Cornford Tony, Smithson Steve [1996]: *Project Research in Information Systems*, A Student's Guide, MacMillan Press Ltd, London

Dahlberg T., Järvinen J. [1997]: Challenges to IS Quality, *Information and Software Technology Journal*, 39 (12): 809-818

Diaz M., Sligo J. [1997]: How Software Process Improvement helped Motorola, *IEEE Software,* 1997, 14, 75-81

Dorling Alec [1993]: Spice: Software Process Improvement and Capability dEtermination, *Software Quality Journal*, 2, 93, pp. 209-224

DTI [1992]: *TickIT making a better job of software, Guide to Software Quality Management System Construction and Certification using ISO9001/EN29001/BS5750* Part 1 (1987): 29 February 1992, Issue 2.0

Fenton, N.E. and Pfleeger, S.L.[1997] 'Rigorous & Practical Approach', PWS Publishing Company

Georgiadou E., Sadler C.[1995] "*Achieving quality improvement through understanding and evaluating Information Systems Development Methodologies*", 3rd. International Conference on Software Quality Management, SQM'95, Seville, Spain

Georgiadou, E. [2003b] "Software Process and Product Improvement A Historical Perspective", International Journal of Cybernetics, January 2003

Georgiadou, E. [2003b] Towards a Customisable, Multi-layered Software Engineering Quality Model, 11th International Conference on Software Quality Management, April 2003, Dundee, Scotland

Georgiadou, E., Keramopoulos, E. [2001a]*"Measuring the Understandability of a Graphical Query Language through a Controlled Experiment*", 9th International Conference on Software Quality Management, SQM 2001, April 2001, University of Loughborough, UK.

Georgiadou, E., Milankovic-Atkinson, M. [1999] *"A formal experiment to verify Object-Oriented Metrics",* *I*NSPIRE'99, Crete, Greece

Georgiadou, E.[2001b] Software Measurement for Process and Product Improvement - Controlled Experiments and Derivation of Reengineering Metrics, Transfer Report (from M.Phil. to PhD.), chool of Informatics and

Multimedia Technology, Faculty of Science Computing and Engineering, University of North London, August 2001

Ginzberg M.J. [1981]: Key Recurrent Issues in the MIS Implementation Process, *MIS Quarterly* 5: 47-59

Grady R. B. [1997]: *Successful Software Process Improvement*, Prentice Hall PTR, Upper Saddle River, NJ

Haase Volkmar H. [1992]: Bootstrap - Measuring Software Management Capabilities, First Findings in Europe, Proceedings of *the fourth IFAC/IFIP Workshop*, Austria, May 92

Haase Volkmar, Messnarz Richard [1994]: Bootstrap: Fine-Tuning Process Assessment IEEE Software, July 1994, pp. 25-35, 1994

Herbsleb James, Carleton Anita, Rozum James, Siegel Jane, Zubrow David [1994]: Benefits of CMM-Based Software Process Improvement: Initial Results, Technical Report, CMU/SEI-94-TR-13, August 1994

Holcombe, M. and Ipate, F.:[1998] [2001] "Correct Systems - Building a Business Process Solution", Springer-Verlag, 1998

Holcombe, M., Bogdanov, K., Gheorghe, M.: “Functional test set generation for extreme programming”, In the Proc. of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering [XP2001], pp. 109-113, Sardinia, Italy, 20-23 May, 2001

Humphrey W. [1995]: *A Discipline for Software Engineering*, Addison Wesley, Reading, MA, USA

Huotari J. & Kaipala, J.:[1999] “Review of HCI Research - Focus on cognitive aspects and used methods”, in the Proc. of Kakola T. [ed.] IRIS 22 Conference: Enterprise Architectures for Virtual Organisations, Keurusselka, Jyvaskyla University Printing House, Finland, 1999

Jackson, M. and Zave, P.:[1993] "Domain descriptions", in Proceedings of the 1st International Symposium on Requirements Engineering, San Diego, Ca, pp. 56-64, 1993

Järvinen J. [1994]: On comparing process assessment results: BOOTSTRAP and CMM, *Software Quality Management, SQM94*, Edinburgh, pp. 247-261

Kautz K. [1999]: Software Process Improvement in Very Small Enterprises: Does It Pay Off? *Software Process – Improvement and Practices,* 4 [4]: , 209-226

Kitchenham, Barbara, and Shari Lawrence Pfleeger.[1996] "Software Quality: The Elusive Target." *IEEE Software* 13, 1, Jan. 1996: 12-21

Klein H, K., Lyytinen K. [1985]: The Poverty of Scientism in Information Systems. In Mumford E. Hirschheim R. Fitzgerald G, Wood-Harper T. [eds.]: *Research Methods in Information Systems,* Proceedings of IFIP WG 8.2 Colloquium, Manchester Business School, 1-3 Sept. 1984, pp. 131-161, North-Holland, Amsterdam

Kuvaja P., Similä J., Kranik L., Bicego A., Saukkonen S., Koch G. [1994]: *Software Process Assessment and Improvement – The BOOTSTRAP Approach*, Blackwell Publishers, Cambridge, MA

Kuvaja Pasi [1999]: New Developments in Software Process Improvement Keynote Speech in *Software Quality Management Conference* [SQM 99]: Southampton March 1999

Kuvaja, 1999] and SPICE [Dorling, 1993; Rout, 1995] concentrate on the process instead of the product by ensuring a disciplined and controlled software development process via independent evaluation.

Logothetis, N. andWynn, H.P., [1989] ‘Quality Through Design: Experimental Design, ‘Off-line Quality Control and Taguchi’s Contributions’, Oxford Science Publications, 1989.

Lucas H.C. Jr. [11981]: *Implementation: The Key to Successful Information* Systems, Columbia University Press, New York

Marcus M.L. [1981]: Implementation Politics: Top Management Support and User Involvement, *Systems, Objectives, Solutions* 1: 203-215

Paulk Mark C. [1993]: Comparing ISO 9001 and Capability Maturity Model for Software, Software Quality Journal 2, 1993, pp. 245 - 256

Paulk Mark C. [1995]: The Rational Planning of [Software]: Projects, *Proceedings of the First World Congress for Software Quality, ASQC,* San Francisco, CA, 20-22 June 1995

Paulk Mark C., Curtis Bill, Chrissis Mary Beth [1993]: Capability Maturity Model, Version 1.1, *IEEE Software*, July 1993, pp. 19-27

Rodenbach E., Van Latum F., Van Solingen R. [2000]: SPI – A Guarantee for Success? – A Reality Story from Industry, *PROFES 2000*, Oulu, Finland, Springer, 216-231

Rout Terence P. [1995]: SPICE: A Framework for Software Process Assessment, *Software Process-Improvement and Practice*, Pilot Issue, pp. 57 – 66

Shepperd, M.J.[1995] *Foundations of Software Measurement* Prentice-Hall: Hemel Hempstead, England

Siakas Kerstin [2002]: SQM-CODES; Software Quality Management – Cultural and Organisational Diversity Evaluation, PhD thesis submitted Nov. 2002 for the degree of PhD at London Metropolitan University

Siakas Kerstin V. and Balstrup Bo [2000]: A field-study of Cultural Influences on Software Process Improvement in a Global Organisation*, European Software Process Improvement Conference, EuroSPI '00*, Copenhagen 7-9 Nov. 2000

Stelzer D., Mellis W. [1998]: Success Factors of Organisational Change in Software Process Improvement, *Software Process – Improvement and Practice*, 4, 227-250

Zahran S. [1998]: *Software Process Improvement: Practical Guidelines for Business Success,* Addison-Wesley Pub. Co. Reading, Mass.

7 Author CVs

Elli Georgiadou

Elli Georgiadou is a Principal Lecturer in Software Engineering at Middlesex University, London. Her teaching includes Software Metrics, Methodologies, CASE and Project Management. She is engaged in research in Software Measurement for Product and Process Improvement, Methodologies, Metamodelling, Cultural Issues and Software Quality Management. She is a member of the University's Global Campus project (developing and offering ODL). She has extensive experience in academia and industry, and has been active in organising/chairing conferences and workshops under the auspices of the British Computer Society, the ACM British Chapter and various European programmes for Technology Transfer and development of joint curricula. She established a Distance Mode Initiative between a UK University and a Hong Kong Institute developing and offering technology-based learning. She has engaged in developing the pedagogic framework as well as the development of materials. She designed and carried out evaluations of various ODL initiatives in the UK, Greece, Spain, Finland, Hong Kong and Cyprus.

Kerstin V. Siakas

Kerstin works as a lecturer since 1989 at the department of Informatics at the Technological Education Institution of Thessaloniki, Greece. She is born and grown up in the Swedish part of Finland. She has an extensive industrial experience (since 1975) in software development on different levels from many European countries and mainly from multinational organisations. Because of her multicultural background and her work experience her research interest is in Software Process Improvement and how culture influences software development. She finished her PhD in November 2002. She has published around 20 papers about her research.

Eleni Berki

Eleni Berki is an Asst. Professor of Group Technologies in Jyväskylä University, Finland. She completed her PhD in Process Metamodelling and Systems Method Engineering in 2001, in United Kingdom. Her teaching and research interests include: Process Metamodelling and Information Systems Engineering, Computational Models and Multidisciplinary Approaches for Software Engineering, Knowledge Representation Frameworks and Requirements Engineering. She has worked as a software designer and consultant in industry, and has a number of academic and industrial project partners in many countries. She has been active in the development, delivery and coordination of virtual and distance learning initiatives in collaboration projects in European and in Asian countries.