In most disciplines, the evolution of knowledge involves learning by observing, formulating theories, and experimenting. Theory formulation represents the encapsulation of knowledge and experience. It is used to create and communicate our basic understanding of the discipline. Checking that our understanding is correct involves testing our theories, i.e., experimentation in some form. Analysing the results of the experimental study promotes learning and the ability to change and refine our theories. These steps take time which is why the understanding of a discipline, and its research methods, evolves over time.

The paradigm of encapsulation of knowledge into theories and the validation and verification of those theories based upon experimentation, empirical evidence, and experience is used in many fields, e.g., physics, medicine, manufacturing.

What do these fields have in common? They evolved as disciplines when they began learning by applying the cycle of observation, theory formulation, and experimentation. In most cases, they began with observation and the recording of what was observed in theories or specific models. They then evolved to manipulating the variables and studying the effects of change in the variables.

How does the paradigm differ for these fields? The differences lie in the objects they study, the properties of those objects, the properties of the systems that contain them and the relationship of the objects to the systems. So differences exist in how the theories are formulated, how models are built, and how studies are performed; often affecting the details of the research methods.

Software engineering has things in common with each of these other disciplines and several differences.

In physics, there are theorists and experimentalists. The discipline has progressed because of the interplay between both groups. Theorists build models (to explain the universe). These models predict the results of events that can be measured. The models may be based upon theory from understanding the essential variables and their interaction or data from prior experiments, or better yet, from both. Experimentalists observe and measure, i.e., carry out studies to test or disprove a theory or to explore a new domain. But at whatever point the cycle is entered there is a pattern of modelling, experimenting, learning and remodelling.

The early Greek model of science was that observation, followed by logical thought, was sufficient for understanding. It took Galileo, and his dropping of balls off the tower at Pisa, to demonstrate the value of experimentation. Eddington’s study of the 1919 eclipse differentiated the domain of applicability of Einstein’s theories vs. Newton’s.

In medicine, we have researchers and practitioners. The researcher aims at understanding the workings of the human body and the effects of various variables, e.g., procedures and drugs. The practitioner aims at applying that knowledge by manipulating those variables for some purpose, e.g., curing an illness. There is a clear relationship between the two; knowledge is often built by feedback from the practitioner to the researcher.

Medicine began as an art form. It evolved as a field when it began observation and theory formulation. For example, Harvey’s controversial theory about the circulation of blood through the body was the result of many careful experiments performed while he practiced medicine in London. Experimentation varies from
controlled experiments to qualitative analysis. Depending on the area of interest, data may be hard to acquire. Human variance causes problems in interpreting results. However, our knowledge of the human body has evolved over time.

The focus in manufacturing is to better understand and control the relationship between process and product for quality control. The nature of the discipline is that the same product is generated, over and over, based upon a set of processes, allowing the building of models with small tolerances. Manufacturing made tremendous strides in improving productivity and quality when it began to focus on observing, model building, and experimenting with variations in the process, measuring its effect on the revised product, building models of what was learned.

The Empirical Software Engineering journal [see the announcements section of this issue of SPN] is dedicated to the position that like other disciplines, software engineering requires the cycle of model building, experimentation, and learning; the belief that software engineering requires empirical study as one of its central concerns. Research has analytic and experimental components. The role of the researcher is to build models of and understand the nature of processes, products, and the relationship between the two in the context of the system in which they live. The practitioner’s role is to build “improved” systems, using the knowledge available and to provide feedback. But like medicine (e.g., Harvey), the distinction between researcher and practitioner is not absolute, some people do both at the same time or at different times in their careers. This mix is especially important in planning empirical studies and when formulating models and theories.

Like manufacturing, these roles are symbiotic. The researcher needs laboratories and they only exist where practitioners build software systems. The practitioner needs to better understand how to build systems more productively and profitably; the researcher can provide the models to help this happen.

Just as the early model of science evolved from learning based purely on logical thought, to learning via experimentation, so must software engineering evolve. It has a similar need to move from simple assertions about the effects of a technique to a scientific discipline based upon observation, theory formulation, and experimentation.

To understand how model building and empirical studies need to be tailored to the discipline, we first need to understand the nature of the discipline. What characterises the software engineering discipline? Software is development not production. Here it is unlike manufacturing. The technologies of the discipline are human based. It is hard to build models and verify them via experiments—as with medicine. As with the other disciplines, there are a large number of variables that cause differences and their effects need to be studied and understood. Currently, there is a lack of models that allow us to reason about the discipline, there is a lack of recognition of the limits of technologies for certain contexts, and there is a lack of analysis and experimentation.

There have been empirical analysis and model building in software engineering but the studies are often isolated events. For example, in one of the earliest empirical studies, Belady & Lehman [3][4] observed the behaviour of OS 360 with respect to releases. They posed several theories that were based upon their observation concerning the entropy of systems. The idea of entropy - that you might redesign a system rather than continue to change it - was a revelation. On the other hand, Basili & Turner [2] observed that a compiler system being developed, using an incremental development approach, failed structure over time. This appears contradictory. But under what conditions is each phenomenon true? What were the variables that caused the different effects? What were the different variables as size, methods, the nature of the changes? We can hypothesise, but what evidence do we have to support those hypotheses?

In another area, Walston and Felix [6] identified 29 variables that had an effect on software productivity in the IBM FSD environment. Boehm [5] observed that 15 variables seemed sufficient to explain/predict the cost of a project across several environments. Bailey and Basili [1] identified 2 composite indexes that were combined with size were a good predictor of effort in the SEL environment. There were many other cost models at the time. Why were the variables different? What did the data tell us about the relationship of variables?

Clearly the answers to these questions require more empirical studies that will allow us to evolve our knowledge of the variables of the discipline and the effects of their interaction. In our discipline, there is little consensus on terminology, often depending upon whether the ancestry of the researcher is the physical sciences, social sciences, medicine, etc. One of the roles of the Empirical Software Engineering journal is to begin to focus on a standard set of definitions.

We tend to use the word experiment broadly, i.e., as a research strategy in which the researcher has control over some of the conditions in which the study takes place and control over the independent variables being studied; an operation carried out under controlled conditions in order to discover an unknown effect of law, to test or establish a hypotheses, or to illustrate a known law. This term thus includes quasi-experiments and pre-experimental designs. We use the term study to mean an act or operation for the purpose of discovering an unknown effect of law, controlling many extraneous variables, particularly, the empirical component is at a very primitive stage in its development. We are learning how to build models, how to design experiments, how to extrapolate that knowledge. We believe there is a need for all kinds of studies: descriptive, correlational, cause-effect studies, studies on novices and experts, studies performed in a laboratory environment or in real projects, quantitative and qualitative studies, and replicated studies.

As a discipline software engineering, and more particularly, the empirical component is at a very primitive stage in its development. We are learning how to build models, how to design experiments, how to extract useful knowledge from experiments, and how to extrapolate that knowledge. We believe there is a need for all kinds of studies: descriptive, correlational, cause-effect studies, studies on novices and experts, studies performed in a laboratory environment or in real projects, quantitative and qualitative studies, and replicated studies.

We would expect that over time, we will see a maturing of the empirical component of software engineering. The level of sophistication of the goals of an experiment and our ability to understand interesting things about the discipline will evolve over time. We would like to see a pattern of knowledge building from series of experiments; researchers building on each others’ work, combining experimental results; studies replicated under similar and differing conditions.

The Empirical Software Engineering journal is a forum for that learning process. Our experiments in some cases, like those in the early stages of other disciplines,
will be primitive. They will have both internal and external validity problems. Some of these problems will be based upon the nature of the disciplines, affecting our ability to generate effective models or effective laboratory environments. These problems will always be with us, as they are with any discipline as it evolves and learns about itself. Some problems will be based on our immaturity in understanding experimentation as a discipline, e.g., not choosing the best possible experimental design, not choosing the best way to analyse the data. But we can learn from weakly designed experiments how to design them better. We can learn how to better analyse the data. The ESE journal encourages people to discuss the weaknesses in their experiments. We encourage authors to provide their data to the journal so that other researchers may re-analyse them.

The journal supports the publication of artifacts and laboratory manuals. For example, in issue 1(2), the paper "The Empirical Investigation of Perspective-based Reading" has associated with it a laboratory manual that will be furnished as part of the ftp site at Kluwer Academic Publishers. It contains everything needed to replicate the experiment, including both the artifacts used and the procedures for analysis. It is hoped that the papers in this journal will reflect successes and failures in experimentation; they will display the problems and attempts at learning how to do things better. At this stage we hope to be open and support the evolution of the experimental discipline in software engineering.

We ask researchers to critique their own experiments and we ask reviewers to evaluate experiments in the context of the current state of the discipline. Remember, that because of the youth of the experimental side of our disciplines, our expectations cannot yet be the same as those of the more mature disciplines, such as physics and medicine.

This goal of the journal is to contribute to a better scientific and engineering basis for software engineering.

References


This article appeared as an editorial to issue 1(2) of the Empirical Software Engineering journal. It is reprinted here with permission of the publisher and author.

Victor Basili can be reached at: Department of Computer Science, University of Maryland, College Park, MD 20742, USA; Email: basili@cs.umd.edu

BENEFITS AND PREREQUISITES OF ISO 9000 BASED SOFTWARE QUALITY MANAGEMENT

Dirk Steizler, Mark Reibnitz, Werner Mellis
University of Koeln

The ISO 9000 quality standards were released a decade ago. Since then thousands of software companies have implemented ISO 9000 based quality systems. In Europe, the ISO 9000 standards are the prevalent model for implementing software quality management. However, only a few empirical studies on ISO 9000 based quality management have been published [1][8][16][23][29][33]. Little is known about benefits software companies have achieved with the help of ISO 9000 based quality systems. Furthermore, a profound knowledge of the enabling and inhibiting factors, e.g. the prerequisites of successful software quality management, is still lacking.

The objective of this paper is (1) to describe benefits that software companies have achieved by implementing ISO 9000 based quality systems, and (2) to identify prerequisites of conducting successful software quality management initiatives.

Research Method

Between October 1996 and December 1997 we analysed published experience reports of 25 software organisations that had implemented an ISO 9000 quality system and sought certification. By examining the experience reports we identified benefits and prerequisites of implementing ISO 9000 based quality management initiatives in software companies.

The study covers experience reports of 12 organisations located in the UK, eight German organisations, two French organisations, and one organisation each in Austria, Greece, and the US. The study includes published reports of software organisations at ACT Financial Systems Ltd. [3], ALCATEL TELECOM [5], ALLDATA [20], ANSWERS Software Service [37], AYX Ltd. [34], BR Business Systems [13], Bull AG [25, 26], Cap Gemini Sogeti [31], CMS (British Steel) [14], Danet-IS GmbH [2, 21], Dr. Materna GmbH [32], IBM Deutschland [6], IDC-UK [28], INTRASOFT [11], Logica [9, 10], Oracle [35], Praxis [15], PSI AG [38], SAP AG [7, 36], Siemens AG [39, 40], Sybase [24], TMBit Software GmbH [30], Triad Special Systems Ltd. [12], Unisys Systems and Technology Operations [4], and an anonymous British software company [27].

The authors of the experience reports are quality managers or senior managers of the software companies. At the time the experience reports were written, the size of the companies ranged from 10 to 2700 employees (mean: 674 employees). The time needed to implement the quality systems ranged from 10 to 96 months (mean: 21 months). The time between certification of the quality system and the publication of the experience reports ranged from 0 to 60 months (mean: 36 months); 72% of the companies had gathered experience with the quality system for more than two years.

Benefits

In the following section we will describe benefits that the authors of the experience reports have attributed to the implementation of ISO 9000 based quality systems.
Twenty three (of 25) reports explicitly describe benefits that were achieved by implementing the quality system. We have summarised these benefits in seven categories: improved project management, improved productivity and efficiency, improved customer satisfaction, improved product quality, more on-time deliveries, positive return on the investment in software quality management, and improved corporate profitability.

Figure 1 shows the percentage of reports that mention benefits relating to each of the categories.

Improved project management is reported by 74 % of the companies. This usually results from better documentation of the software process and from improved communication among staff members and managers in different organisational units of the company. ISO 9000 based quality systems lead to better visibility of the software process, improved documents and checklists, clearer definition of responsibilities, and making use of experience and best practices of other projects. Improved project management leads to a variety of other benefits:

• 48 % of the companies report improved productivity and efficiency of software development.
• 43 % of the companies report improved customer satisfaction.
• 43 % of the companies report improved product quality (usually described in a reduction of defects delivered to customers).
• 17 % of the companies report more on-time deliveries.

26 % of the companies explicitly mention a positive return on the investment in software quality management. 13 % of the companies report improved corporate profitability that they attribute to the implementation of ISO 9000 based software quality management.

However, only 6 out of 23 companies (26 %) support their statements on benefits with quantitative data. Examples are a reduction of budget overruns by 50 % in 4 years [31], a reduction of 13 % in post-installation support costs [13], a reduction of programmers' time spent for hotline support by a factor of 3 [27], and a reduction of overall software development cost by 20 % [38]. The other 17 companies that address benefits of ISO 9000 based quality management do not give any quantitative data. Presumably, the statements on benefits in these reports primarily reflect perceived advantages of implementing quality systems.

Prerequisites

The term "prerequisites" summarises factors that authors of the experience reports covered in our study regard as essential when implementing an ISO 9000 based quality system. Implementation(219,566),(847,865) of the factors has facilitated the success of software quality management; a lack of compliance with the factors has delayed progress in quality management or made it difficult to achieve.

<table>
<thead>
<tr>
<th>Prerequisites of successful software quality management</th>
<th>Percentage of experience reports addressing the factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management commitment and support</td>
<td>84%</td>
</tr>
<tr>
<td>Staff involvement</td>
<td>84%</td>
</tr>
<tr>
<td>Providing enhanced understanding</td>
<td>72%</td>
</tr>
<tr>
<td>Tailoring improvement initiatives</td>
<td>68%</td>
</tr>
<tr>
<td>Encouraging communication and collaboration</td>
<td>64%</td>
</tr>
<tr>
<td>Managing the improvement project</td>
<td>56%</td>
</tr>
<tr>
<td>Change agents and opinion leaders</td>
<td>52%</td>
</tr>
<tr>
<td>Stabilizing changed processes</td>
<td>52%</td>
</tr>
<tr>
<td>Setting relevant and realistic objectives</td>
<td>44%</td>
</tr>
<tr>
<td>Unfreezing the organization</td>
<td>24%</td>
</tr>
</tbody>
</table>

Figure 2: Prerequisites of successful software quality management (n=25)
We identified 10 prerequisites of successful software quality management efforts. Figure 2 shows the factors and the percentage of experience reports addressing these factors.

Management commitment and support is the degree to which which management at all organisational levels sponsor the implementation of the quality system. The necessary investment of time, money, and effort and the need to overcome staff resistance are potential impediments to successful ISO 9000 based improvement initiatives. These obstacles cannot be overcome without management commitment. Active participation and visible support of senior management may give the necessary momentum to the initiative. This positively influences the success of the quality system. 84 % of the experience reports emphasise the importance of management commitment and support.

Staff involvement is the degree to which staff members participate in quality management activities. Staff involvement is essential to avoid a schism between software engineers in development projects and quality managers responsible for implementing the quality system. Staff involvement is also necessary to develop knowledge and understanding of the current processes. Using the skills and experience of employees guarantees that the resulting quality system is a consensus that reflects the practical considerations of diverse projects. 84 % of the authors address this point.

Providing enhanced understanding to managers and staff members comprises acquiring and transferring knowledge of current practices. Managers usually have a general idea of the software process, but they do not have complete understanding of essential details. Employees often do not understand how their work contributes to the corporate mission and vision. Successful quality management initiatives give managers a clearer picture of current practices and they give staff members the opportunity to better understand the business of their organisation. 72 % of the authors emphasise the significance of this topic.

Tailoring improvement initiatives means adapting quality management efforts to the specific strengths and weaknesses of different teams and departments in the company. Standardised and centralised quality systems and staff well versed in quality management must clearly and continually demonstrate benefits to projects. Tailoring increases the compatibility of improvement plans with existing values, past experience, and needs of various projects within an organisation. Tailoring helps to implement a quality system that responds to the true needs of the organisation. 68 % of all reports stress this point.

Successful quality management initiatives have encouraged communication among staff members. This has helped to rectify rumors, to preclude misunderstandings, and to overcome resistance of staff members. Successful quality management efforts have also emphasised collaboration of different teams and divisions. Close cooperation of organisational units provides natural feedback loops, enhances staff members’ understanding and knowledge, encourages people to exploit synergy, and consequently improves productivity and quality. Intensive communication and collaboration help to create a coherent organisational culture that is necessary for achieving substantial improvements. 64 % of the authors mention the importance of communication and collaboration.

Managing the improvement project means that the implementation of the ISO 9000 based quality system is treated like a professional project. At the beginning, in some organisations the quality management projects had neither specified requirements nor had they elaborated a formal project plan, defined milestones, or outlined a schedule. Areas of responsibility were not accurately defined, and the initiative was lacking effective interfaces between quality management and software development teams. Successful initiatives set up and ran the quality management project like a software development project. They used existing project management standards, analysing the differences of defining explicit objectives, established milestones, and monitored progress. 56 % of the reports address this factor.

Change agents are individuals or teams external to the system that is to be improved. Quality managers or consultants usually play the role of change agents. Often, they initiate the quality projects, request resources, and encourage local improvement efforts. They also provide technical support and feedback, publish successes, and keep staff members aware of the quality management efforts. Opinion leaders are members of a social system in which they exert influence. Experienced project managers or proficient software engineers usually act as opinion leaders. They are indispensable for overcoming the potential schism between software development and quality management. They help to tailor the improvement suggestions to the needs of different teams and organisational units. 52 % of the authors mention this issue.

Stabilising changed processes means continually supporting maintenance and improvement of the quality system at a local level. Staff members adopting new activities need continuous feedback, motivation, reinforcement, and guidance to stay involved in the improvement effort. They also need guidance and support to overcome initial problems and difficulties. Stabilising changed processes prevents that improved performance slides back to the old level. 52 % of the reports emphasise the need to stabilise changed processes.

Setting relevant objectives means that the quality management efforts attempt to contribute to the success of the organisation. Mere conformance to the standards or attaining certification usually is not a relevant goal for staff members. It is essential that staff members understand the relationship between quality management and business objectives of the organisation. Setting realistic objectives means that the goals may be achieved in the foreseeable future and with reasonable amount of resources. 44 % of the authors address this point.

Lewin [22] has introduced the importance of "unfreezing" organisations before substantial improvements can be achieved. He emphasises that social processes usually have an "inner resistance" to change. To overcome this resistance an additional force is required, a force sufficient to break the habit and to unfreeze the custom. In software companies that have successfully implemented ISO 9000 based software quality systems perceived deficiencies in software development, management commitment, and competitive pressure have contributed to unfreezing the organisation. 24 % of the reports mention this topic.

Discussion

The findings of our study are based on experience reports written by managers of software organisations. Of course these sources primarily reflect the personal views of the authors of the reports. Nevertheless, the findings give interesting insights into benefits software companies might achieve and factors they should consider when implementing ISO 9000 based software quality management.
Benefits

One would expect that quality managers will tend to publish their experiences with software quality management if the improvement efforts have been successful. Experiences with less successful quality systems are less likely to be published. Therefore, our findings may be biased because we analysed published experience reports only. Presumably, representative studies covering successful and unsuccessful quality systems would reveal lower percentages of companies addressing benefits.

We have summarised the benefits described in the experience reports in seven categories. Improved project management is the prevailing benefit category. In most companies it leads to higher productivity and efficiency, improved customer satisfaction, or to improved product quality. However, more on-time deliveries are reported only by 4 of 23 companies. In the majority of the companies the implementation of ISO 9001 based quality systems does obviously not help to meet schedule commitments more often. This is astonishing since the ability to meet schedules more often is usually one of the benefits expected from the implementation of quality systems.

Twenty three of 25 experience reports explicitly mention at least one benefit of the quality system. Surprisingly, only 6 of 23 companies report a positive return on investment in software quality management, and only 3 of 23 companies mention improved corporate profitability. This might lead to the conclusion that the majority of the companies have not achieved a positive return on investment in software quality management. One might also conclude that ISO 9001 based quality systems will usually not improve corporate profitability. However, not a single experience report explicitly mentions that the quality efforts have not produced a positive return on investment or have not lead to a higher profitability.

It is more likely that the small number of companies reporting positive returns on investment and improved profitability can be put down to the fact that most companies do not conduct comprehensive measurements of costs and benefits of software quality management.

Only 26% of the companies support their statements on benefits of ISO 9000 based quality management with quantitative data. This is remarkable because ISO 9001 [18] and ISO 9000-3 [17] suggest to use measurement and statistical techniques to establish, control and verify process capability. Furthermore, clause 4.4.1 (quality policy) states: "It is important that the effectiveness of a quality system be measured in financial terms". However, a previous empirical study [33] has already shown that many software companies ignore the suggestions of the ISO 9000 standards to conduct measurements.

Prerequisites

Surprisingly, only 3 of the 10 prerequisites identified in our study are explicitly mentioned in ISO 9001: management commitment and support in clause 4.1 (management responsibility), managing the improvement project in clause 4.2 (quality system), and setting relevant and realistic objectives in clause 4.1.1 (quality policy) of ISO 9001. This means that companies that strictly stick to the elements of ISO 9001 will probably ignore other essential prerequisites of successful software quality management. Therefore, it is necessary to implement a more comprehensive approach to achieve substantial improvements.

Most of the prerequisites identified in our study address the management of change, that is transforming assumptions, habits, and working routines of managers and staff members so that the quality system may become effective. Implementing ISO 9000 based software quality management requires their changes to an organisation. Our study has shown that many software companies have obviously underestimated the effort needed to accomplish the change process. This indicates that change management is not sufficiently accounted for in the ISO 9000 standards.

At first glance, the prerequisites discussed in this paper may be taken for granted. At least, they seem to be basics of software management. However, when one looks at the experience reports a second time it becomes clear that the factors are regularly described as lessons learned. Some organisations have obviously not paid enough attention to the implementation of the factors at the beginning of the initiative. Other organisations may not have fully understood the significance of the prerequisites until the improvement objectives had been accomplished. Obviously, most quality managers do not pay sufficient attention to the management of change when implementing ISO 9000 based software quality systems.

Concluding Remarks

Most software companies achieve benefits with the implementation of ISO 9000 based quality systems. Only few companies, however, report a positive return on the investment in software quality management and improved corporate profitability. This might lead to the conclusion that the majority of the companies have not achieved a positive return on investment and improved corporate profitability. However, one might also conclude that the small number of companies reporting economic success can be put down to the fact that most companies do not conduct comprehensive measurements of costs and benefits of software quality management.

Most of the prerequisites of successful software quality management identified in our study address the management of change. The fact that the authors of experience reports emphasise these prerequisites as lessons learned shows that the factors are obviously not sufficiently accounted for in the ISO 9000 standards. Change management should therefore be a central element of future versions of the ISO 9000 family.

References

[4] A. Clarke, "Persuading the Staff or ISO 9001 without Tantrums". In SQM, no. 9, pp. 1-5.
**THE PERSONAL SOFTWARE PROCESS AS A CONTEXT FOR EMPIRICAL STUDIES**

Claes Wohlin
Lund University

This article discusses the use of the Personal Software Process (PSP) as a context for doing empirical studies. It is argued that the PSP provides an interesting environment for doing empirical studies. In particular, if we already teach or use the PSP then it could be wise to also conduct empirical studies as part of that effort. The objective of this paper is to present the idea and discuss the opportunities in combining the PSP with empirical studies. Two empirical studies, one experiment and one case study, are presented to illustrate the idea. It is concluded that we obtain some new and interesting opportunities, in particular we obtain a well-defined context and hence ease replication of the empirical studies considerably.
Introduction

Different decades have different trends in software engineering. In the 90’s, we have seen a strong focus on the process and also on the use of empirical methods in software engineering. The need for a scientific approach to software engineering has been stressed, see for example [5]. In this article, we would like to look at the opportunity to combine two of the trends in the 90’s, i.e., process focus and empirical studies.

The objective of the article is to highlight and discuss the opportunities of performing empirical studies within the context of the Personal Software Process (PSP). The PSP is a well-defined process and the process is publicly available through the book by Humphrey [9]. This means that studies conducted within this context can be replicated rather easily, which is critical to the success of applying empirical methods in software engineering.

The article is organized as follows. In the next section, the PSP is briefly discussed. This is followed by a brief introduction to empirical studies. Subsequently, the ability to use the PSP as a context for empirical studies is discussed. The advantages, challenges and opportunities are outlined. Then, two examples of studies conducted with the PSP as an empirical context are presented to illustrate the approach. Finally, the paper is concluded.

The Personal Software Process

The Personal Software Process (PSP) has gained lot of attention since it became publicly available [9]. The objective of the PSP is basically to provide a structured and systematic way for individuals to control and improve their way of developing software. We have seen papers, for example [10], presenting the outcome of the PSP experiments. The PSP is currently used in a number of universities and industry is also becoming interested in applying the PSP.

At Lund University, we run the PSP as an optional course for students in the Computer Science and Technology program and the Electrical Engineering program. Most students take the course in their fourth year, and the course is taken by 50-70 students. The course is run for the second time during the autumn of 1997. The main objective of the course is to teach the students the use of planning, measurement, estimation, postmortem analysis and systematic reuse of experiences. It is from a course perspective more important to teach the students the techniques packaged within the PSP than actually teaching them the PSP for future use.

Empirical Research

Another area which attracts attention is the use of empirical methods in software engineering. The need for experimentation was stressed already in the 1980s [1], but it is during the last couple of years that we have seen a stronger focus on the use of empirical methods, for example, emphasized in [5]. Experiments and case studies will allow us to gain a better understanding of relationships in software engineering and they will also allow us to evaluate different hypotheses. Numerous books on design and analysis of experiments in general are available, for example [13], and case studies are, for example, discussed in [14].

One major difficulty in empirical studies is the validity of the results. In other words how do we interpret the results and what conclusions can we draw? A particular problem is, of course, to find suitable subjects (participants in the experiment). It is desirable to use industrial software engineers, but many times this is unfeasible. A suitable starting point is therefore many times to start with experiments at the universities using students as subjects, i.e. the experiments are conducted in an educational context. The use of students as subjects is, of course, a problem to the validity, but on the other hand it could be good to start with an experiment in a university setting and based on the outcome we can, as part of a technology transfer process, replicate the experiment in industry and then continue with a pilot project.

Experiments are regularly conducted with students as subjects, see for example [2]. The experiments are done as part of a course in software engineering or as a separate activity where the students are attracted to the experiment based on some reward. Thus, we often have to resort to using the students and hence we would like to raise the question: is it possible to use the PSP as a context for empirical studies, including both experiments and case studies? If we use the PSP in the education could we also use the PSP course as a means for empirical studies?

The PSP and Empirical Studies

The PSP can be viewed from two different perspectives when it comes to empirical studies. First, it is important to evaluate the effect of the PSP. This means that the PSP is the object of the study. We have seen results published [10], but further studies are needed. This includes both reports on the outcome from taking the PSP as a course and from industrial use of the PSP. It is, however, not the intention here to discuss this matter. Second, assuming that we have started to teach and use the PSP, the PSP can be used as a context for empirical studies and hence as a vehicle for evaluating different methods and techniques, and to study different relationships in software engineering.

The second issue is the main objective of this article. In particular, the objective is to highlight the opportunities and limitations of using the PSP as a context for empirical studies. Two different types of empirical studies can be identified:

- Experiments aimed at testing a hypothesis, for example, comparing two different methods for inspections. For this type of studies we apply methods for statistical inference, and we would like to show with statistical significance that one method is better than the other [13].

- Case studies used to build models relating attributes to each other, for example, prediction models. One example of a prediction model may be to predict the number of faults in testing based on the number of defects found in compilation and the time spent in inspections. We mainly apply multivariate statistical analysis in this type of studies. The analysis methods include linear regression and principal components analysis [12].

Both of these types of studies are briefly illustrated within the context of the PSP below. It should be noted that the main objective is to illustrate empirical studies using the PSP rather than presenting the studies in detail.

A key question to consider is, of course, why should we use the PSP as a context for empirical studies? Or we could negate the question, and ask why should we not use the PSP as a context for empirical studies? We would like to argue, since we often have to resort to using students in experiments anyway, that it is suitable to use the PSP (at least if we teach the PSP independent of our interest in experimentation). The PSP provides a context which includes collection of several measures which are valuable when experimenting. Moreover, we also believe that the PSP can be the starting point for
The validity of the observations and findings is crucial. The context is given by the definition of the PSP as described by Humphrey [9]. We may want to change the proposed PSP slightly, but basically the context is provided, and hence we do not have to define the context and describe it very carefully to allow for others to understand our study from the context perspective.

Replication. The context also forms the basis for replication. A major problem in empirical studies is that in order to come up with generally valid observations, we must be able to perform a study several times to build up general experience. Thus, the PSP may be one way to ease replication. In other words, experiments and case studies can be conducted at several places using the PSP simultaneously. The PSP provides a stable process and the process description is generally available.

Measures. This is also closely related to the PSP. Measures are collected as an integrated part of the PSP, and it is fairly easy to add measures of specific interest for an empirical study. Thus, the PSP provides a good starting point for collecting measures to use for hypothesis testing and model building.

Challenges

Scaling. The PSP implements activities performed in large-scale projects, hence scaling down, for example, planning and estimation to the individual level. A major challenge in empirical studies is to generalize, see also validity, the observations. The major challenge in using the PSP as context is the ability to scale the observation to other environments, and in particular to large-scale software development. On the one hand, it is difficult to scale individual results to large project. On the other hand the PSP is supposed to act as a down-scaled project.

Validity. The validity of the observations and findings is crucial. We would like to be able to generalize the observations. In order to do this, we must consider different types of validity, for example internal and external validity, [3]. The actual validity for different studies must be addressed separately, as the ability is highly dependent on the study and what we intend to generalize.

Opportunities

The PSP provides some opportunities for empirical studies. We may study the use of different techniques and methods, and investigate the relationships between different attributes. The main limitation of using the PSP as a basis for empirical studies is that we cannot use it to study group activities. It is possible to experiment with using different reading techniques on an individual basis, but we are unable to study the use of inspections and group meetings. Thus, we cannot expect to use the PSP as a context for experimentation for all types of empirical studies, but it is our firm belief that it opens some new opportunities and that the major inhibiting factor is our imagination.

It is clear that the PSP provides opportunities for empirical studies. In the following section, we will show two examples of studies conducted with the PSP as a context. The main objective of the examples is to illustrate the use of the PSP as context, and not to provide any deep insight into the actual empirical studies.

Illustration of Using the PSP for Empirical Studies

Introduction

The main objective of this section is to illustrate the use of the PSP as a context for empirical studies. The actual results of the studies are presented here, but not in full detail.

The objectives of the empirical studies are:

• to evaluate the difference in fault density based on prior experience of the programming language, and
• to investigate the relationships between different performance measures.

For the first case, we have collected background information on the experience of the programming language used. In our particular case, the students should use C as the programming language independent of their prior experience. We used randomization [4]. In a case study, we do not expect to have randomization. In general, we have much less control in a case study than in an experiment.

Planning

As part of the first lecture, the students were asked to fill out a survey regarding their background in terms of experiences from issues related to the course, for example, knowledge in C. The students were required to use C in the course independently of their prior experience of the language. Thus, we did not require that the students had taken a C-course prior to entering the PSP course, which meant that some students learnt C within the PSP course. This is not according to the recommendation by Humphrey, see [9]. The hypothesis of the experiment based on the C experience was that students with more experience in C would make fewer faults per lines of code.
The case study is based on investigating several performance measures and after the course evaluate if they measure several different dimensions, and in particular if we are able to capture quality, productivity, cycle time and predictability from the performance measures. It should be noted that we use all 10 programming tasks in the PSP course as the basis for determining the performance. The following measures were defined as performance measures: Total number of faults, Fault density, Program size, Development time, Productivity, Predictability of size, and Predictability of time. The objective of the second study does not require anything in particular during the course, since it is primarily an analysis at the end.

For the first case, we have the following hypothesis:

Null hypothesis, H0: There is no difference between the students in terms of number of faults per KLOC (1000 lines of code) based on the prior knowledge in C.

• H0: Number of faults per KLOC is independent of C experience.

• H1: Number of faults per KLOC changes with C experience.

Measures needed: C experience and Faults/KLOC.

The C experience is measured by introducing a classification into four classes based on prior experience of C (ordinal scale). The classes are:

1. No prior experience.
2. Read a book or followed a course.
3. Some industrial experience (less than 6 months).
4. Industrial experience.

The second investigation requires that the following data are collected: Program size (estimate and actual), Development time (estimate and actual), and Number of faults. From these measures, we are able to derive the performance measures. The fault density of Faults/KLOC is also used in the first case. It should be noted that we are unable to evaluate the cycle time as we have no measures which capture the cycle time. This is difficult to achieve within the PSP. The best we can do is probably to measure delivery precision, primarily in terms of the number of late deliveries. We have, however, not kept track of this information or at least to the degree that we trust the data.

The experimental design for language experience is: one factor with more than two treatments. The factor is the experience in C, and we have four treatments, see the experience grading above. The dependent variable is measured on a ratio scale and we can use a parametric test for this hypothesis. The ANOVA test is hence suitable to use for evaluation.

For the case study, we would like to use principal components analysis to study what dimensions we are capturing with our seven performance measures. It is quite common that we collect a large number of measures, but basically we are only capturing a few dimensions due to multicollinearity between the different measures.

Validity Evaluation

This is a difficult area. In our particular case, we have several levels of validity to consider. Internal validity can be divided into: within the course this year, and between years. External validity can be divided into: students at Lund University (or more realistically to students from programs taking the PSP course), the PSP in general, and for software development in general.

The internal validity within the course is probably not a problem. The large number of tests (equal to the number of students) ensures that we have a good internal validity, probably both within the course this year and similar results can be expected in the future, if running the course in a similar way.

Concerning the threats to the external validity, it is difficult to generalize the results to other students, i.e. students not taking the course. They are probably not as interested in software development and hence they come from a different population. The results from the analysis can probably be generalized to other PSP courses, where it is feasible to compare participants based on their background in terms of computer science or electrical engineering or experience of a particular programming language.

The results are found for the PSP, but they are likely to hold for software development in general. This is motivated by the following observations for the two studies:

• There is no reason that people having different background experience from a particular programming language perform differently between the PSP and software development in general. Thus, for the language background experience, we expect similar results in other environments.

• The performance measures can be collected for other environments than the PSP, and there is no reason that we should not get a similar grouping of the different measures. Thus, we believe that the results can be generalized to other contexts.

Operation

The subjects (students) are not aware of what we intend to study. They were informed that we wanted to study the outcome of the PSP course in comparison with the background of the participants. They were, however, not aware of the actual hypotheses stated. The students, from their point of view, do not primarily participate in an empirical study; they are taking a course. All students are guaranteed anonymity.

The survey material is prepared in advance. Most of the other material is, however, provided through the PSP book [9]. The empirical study is executed over 14 weeks, where the 10 programming assignments are handed in regularly. The data are primarily collected through forms. Interviews are used at the end of the course, primarily to evaluate the course and the PSP as such.

Data Validation

Data were collected for 65 students. After the course, the achievements of the students were discussed among the people involved in the course. Data from six students were removed, due to that the data being regarded as invalid or at least questionable. Students have not been removed from the evaluation based on the actual figures, but due to our trust in the delivered data. The six students were removed due to:

• Data from two students were not filled in properly.

• One student finished the course much later than the rest, and he had a long period where he did not work with the PSP. This may have affected the data.

• The data from two students were removed based on the fact that they delivered their assignments late and required considerably more support than the other students, hence it was judged that the extra advice may have affected their data.
Finally, one student was removed based on the fact that his background is completely different from the others. This means removing six students out of the 65, hence leaving 59 students for statistical analysis and interpretation of the results.

### Analysis and Interpretation

#### Analysis of experiment

For the experiment, we use descriptive statistics to visualize the data collected. From plotting the data, it is obvious that we have one outlier. If we look at the data including the outlier in the analysis, it seems that there is a weak tendency (when looking at the mean value) towards that more experience means lower fault density. If we remove the outlier, the tendency is still there although very weak. The data are summarized in Table 1. Thus, we do not expect to find any support for the hypothesis that language experience affects the fault density. 

The next step is to apply an ANOVA test to evaluate the hypothesis that more experience in C means fewer faults/KLOC. The results of the analysis are shown in Table 2.

As expected, the results from the analysis are not significant. Thus, we are unable to show that there is a significant difference in terms of number of faults/KLOC based on C experience. Since the number of students in class 3 and 4 is very limited, class 2, 3 and 4 are grouped together to study the difference between class 1 and a grouping of class 2-4. A t-test was performed to evaluate if it was possible to differentiate between class 1 and the rest. No significant results were obtained.

### Analysis of case study data

For the case study, we are interested to investigate if we are able to capture different dimensions through our seven performance measures. In particular, we would like to see how many dimensions our seven measures actually capture. In order to do this, we apply a principal components analysis. The results of the analysis are presented in Table 3.

From Table 3, we note that the seven measures can be grouped into three factors. The first factor seems to mainly capture faults. The development time is included in this factor, which may be regarded as a surprise, but it can be explained by the argument that a driving factor of time is the number of faults. People who have many faults take longer time to develop their programs, hence supporting the hypothesis that fault prevention and early fault detection is important for the development time. The second factor includes program size and productivity. This result indicates the difficulty we have in capturing productivity, i.e. people who write large programs seems to have a higher productivity. In this case when the students develop the same programs, productivity should be measured in terms of time to implement the functionality rather than as defined here. The third factor clearly captures the ability to estimate accurately, i.e. predictability.

From the case study, we can see that we manage to differentiate between the three factors: quality (in terms of faults), productivity (although a questionable measure) and predictability. The fourth major factor (see the section on Planning), i.e. cycle time, is not measured and hence, of course, not visible among the factors found in the principal component analysis.

### Discussion

We would like to see the Personal Software Process as an opportunity for empirical studies, in addition to its

---

**Table 1**: Faults/KLOC for the different C experience classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Number of Students</th>
<th>Median value of Faults/KLOC</th>
<th>Mean value of Faults/KLOC</th>
<th>Standard deviation of Faults/KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31</td>
<td>66.0</td>
<td>72.7</td>
<td>29.0</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>69.7</td>
<td>68.0</td>
<td>22.9</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>63.6</td>
<td>67.6</td>
<td>20.6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>63.0</td>
<td>63.0</td>
<td>17.3</td>
</tr>
</tbody>
</table>

**Table 2**: Results from the ANOVA test.

<table>
<thead>
<tr>
<th>Performance Measure</th>
<th>Factor 1: Faults</th>
<th>Factor 2: Productivity</th>
<th>Factor 3: Predictability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults</td>
<td>0.869</td>
<td>0.395</td>
<td>0.019</td>
</tr>
<tr>
<td>Faults/KLOC</td>
<td>0.886</td>
<td>0.183</td>
<td>0.177</td>
</tr>
<tr>
<td>Development time</td>
<td>0.815</td>
<td>-0.282</td>
<td>-0.083</td>
</tr>
<tr>
<td>Program size</td>
<td>0.156</td>
<td>0.824</td>
<td>0.064</td>
</tr>
<tr>
<td>Productivity</td>
<td>-0.570</td>
<td>0.778</td>
<td>0.134</td>
</tr>
<tr>
<td>Size predictability</td>
<td>-0.218</td>
<td>-0.141</td>
<td>0.740</td>
</tr>
<tr>
<td>Time predictability</td>
<td>-0.036</td>
<td>-0.247</td>
<td>0.740</td>
</tr>
</tbody>
</table>

**Table 3**: Results from the principal components analysis.
original objective. Empirical studies are an important means to further understand, evaluate and improve software development. The empirical studies are often conducted in a student setting, and if we teach the PSP it may be combined with our need for empirical studies. Furthermore, it may be an important step in technology transfer. Experiments and case studies can be conducted in a controlled environment, before transferring the results to an industrial environment for further studies and implementation in the industrial development processes. It should also be noted that the empirical studies can be replicated easily based on the well-defined context provided by the PSP, hence providing a good basis for technology transfer decisions.

The results from our two studies, briefly outlined in the paper, are interesting in themselves. The experiment shows that the experience of C programming does not influence the fault density. The case study illustrates that although trying to define seven different performance measures, we have actually only captured three main factors. It is also interesting to note that the factors are possible to separate, and that they are in accordance with expectations.

We have already conducted several studies using the PSP as a context for empirical studies, and based on our experience and the results obtained we will continue our research in this direction. The studies so far include: two effort estimation studies (prestudy and main study) [6] [7], a programming language comparison study [8] and a study of the performance in the PSP based on the individual background [15].

Finally, we would like to encourage others to perform empirical studies within the PSP both to replicate our studies and to perform other studies which could enlighten and improve our understanding of the underlying phenomena in software engineering.

Acknowledgment

I would like to thank Per Runeson, Dept. of Communication Systems for valuable comments on a draft of this paper.

References


Claes Wohlin can be reached at: Dept. of Communication Systems, Lund University, PO Box 118, SE-221 00 Lund, Sweden; E-mail: claeswa@tts.lth.se

SPICE TRIALS ASSESSMENT PROFILE

Robin Hunter
University of Strathclyde

This paper summarises the demographic information concerning the data collected in conjunction with Phase 2 of the SPICE Trials1 up until 15th December 1997. Further information about the SPICE Trials and the version of the emerging ISO/IEC 15504 international standard that was evaluated during these trials can be obtained from [1]. We first describe the main demographic factors for which a significant amount of data was collected. Then we summarise the trials in terms of process coverage, summarise the ratings and capability levels observed, present some initial analyses on the impact of criticality on process capability, and then present a summary and conclusions.

Summary of Assessments and Projects

A large amount of demographic information concerning the trials was collected (much more than for Phase 1). Some of this data concerned the Organisational Units (OU’s) that were assessed and some concerned the projects that were assessed within the OUs. In this section we summarise this information.

OU Data

The Organisational Unit data (OU data) included the SPICE region in which the OU was situated, the industrial sector in which the OU operated, the target sector for which the OU produced software, the total number of staff in the OU, and the number of IT staff in the OU.

From Figure 1 it is seen that the assessments were split roughly equally between two of the five SPICE regions, with 16 in Europe and 14 in the Southern Asia Pacific region, giving a total of 30 assessments for which we have data.

The distribution shown in Figure 2 shows that, out of the 30 assessments, 90% (27/30) used the Part 5 assessment model. The remaining 10% used the Process Professional assessment model. Figure 3 shows the distribution of tools used. Most of the assessments (67%) did not use an assessment tool. Of

1 The interim Trials Report is available publicly and can be obtained from <http://www.iese.fhg.de/SPICE> (goto the Trials page) or <http://www.sqi.gu.au/spice/trials.shtml>

Software Process Newsletter: SPN - 12
those that used a tool, 23% (7/30) used the SEAL tool from South Africa (available in [1]), and the remaining 10% (3/30) used the Process Professional assessment tool.

Since more than one assessment may have occurred in a particular OU (for example, multiple assessments, each one looking at a different set of processes), we can

Figure 1: Region where the assessments took place (the y-axis is number of assessments).

Figure 2: Distribution of assessment models used.

Figure 3: Distribution of assessment tools used.

Figure 4: Region where participating OUs are located (y-axis is the number of OUs).

Figure 5: Primary business sector of OUs participating in the trials (y-axis is the number of OUs).

Figure 6: Target business sector of the OUs participating in the trials (y-axis is the number of OUs).

Figure 7: Approximate number of OU staff in participating OUs.

Figure 8: Approximate number of IT staff in participating OUs.
see in Figure 4 that organisations involved in the assessments were split with 15 in Europe and eight in the Southern Asia Pacific region, giving a total of 23 different organisations. Figure 5 shows that 11 of the OUs were concerned with the production of software or other IT products or services. Figure 6 shows the target sectors (one or more) in which each of the OUs were involved.

The data for the approximate number of staff and the approximate number of IT staff in the OUs are shown in Figure 7 and Figure 8 for 21 of the 23 OUs. The questions corresponding to these data both asked for approximate numbers of staff, rounded to a suitable number, 'such as' those shown. It would have been perfectly possible for a number greater than 1000 (in the case of Figure 7) and greater than 500 (in the case of Figure 8) to have been returned, and the database allowed for this. However, no such numbers were returned from the trials.

As can be seen from this data, there was good variation in the sizes (both small and large) of the OUs that participated in the trials thus far. However, the same cannot be said for the business sectors. No organisations in the following primary business sectors participated in the trials (see Figure 5): business services, petroleum, automotive, aerospace, public administration, consumer goods, retail, health and pharmaceuticals, leisure and tourism, manufacturing, construction, and travel.

Project data

More than one project may be assessed in a single assessment. Project specific data we collected included the criticality of the product produced, the perceived importance of the product quality characteristics defined by ISO/IEC 9126, and the category to which the product belonged.

We had data from the 76 projects involved in the trials. Approximately 80% of these were software development projects, approximately 4% were non-software development projects, and approximately 13% were continuous processes within the organisation not associated with a single project.

The number of projects per trial is shown in Figure 9. It is evident that most assessments involved only one project. However, some covered up to 10 projects in a single assessment.

The product categories for these projects are shown in Figure 10. We had data from only 56 of these projects. As can be seen, almost half of the projects involved the development of information systems of one sort or another. Of the information system category, two projects were non-software development. Of the operating system category, three were continuous organisational processes. Of the database management category, one was a continuous organisational process.

The distribution of the projects according to code size is shown in Figure 11, where small means less than 10 KLOC, medium, 10-100 KLOC, and large more than 100 KLOC for a software system implemented in 3GL. The data was available for 26 of the 76 projects. Although it is highly dubious to collect line of code data across organisations internationally in such a manner, it still gives a rough indication of project sizes. Perhaps most interesting is the extent of inability of organizations to provide size data on their project (note that size in Function Points was also requested, but there even less data was collected).

Process Coverage

The process instances assessed during the trials (341 in all) were distributed over the five process categories defined by the ISO/IEC 15504 model (CUS, ENG, SUP, MAN, ORG) as in Figure 12. As can be seen, all the
The ratings of the attributes associated with a process instance may be used to compute the capability of a process. The capability of a process is defined to be the highest capability level for which the process attributes for that level are either rated largely or fully and the attributes for all lower levels are rated fully. A summary of this scheme is provided in Table 2.

When the data in the database is analysed the number of process instances found to be at each of the capability levels is as shown in Figure 17.

A comment on the definition of process capability may be appropriate here. Clearly there are two ways in which a process instance may fail to be rated at a particular capability level:

- The attributes at that level may not be rated fully or largely.
- The attributes at the next lower level may not be rated fully.

The 65 process instances at level 2 were analysed to see which would have been rated at level 3 if this did not require the level 2 attributes to be rated fully rather than largely. The result was that 24 process instances would have been rated at level 3 (37%). Thus in a significant number of cases, process instances fail to achieve a particular capability level because of inadequacies at the previous level, rather than at the level in question.

When performing the above analysis, one anomaly was noticed, namely a process which did not fully satisfy the level 2 attributes and yet fully satisfied the level 3 attributes. The rest of the data suggested that this was an isolated case!

The numbers of process instances at each capability level may also be shown for each process category as in Figure 18, which shows the percentage of process instances in each category achieving at least a particular capability level. Notice that the process instances at level 0 tend to be in the SUP and MAN categories, while the level 4 process instances tend to be in the SUP category.

### Criticality

Clearly, as can be seen from the previous sections, there is considerable scope for correlating demographic variables with process ratings. As an example of this, an analysis was performed of how the criticality factors concerning safety, economic loss, security, and environmental impact (as defined in ISO/IEC 14598) affected the process ratings. The capability levels of those process instances associated with projects that the OU considered to be critical with respect to one of the factors (a subset containing 88 of the 341 process instances considered above) are summarised in Figure 19. Most notable are the smaller percentage of level 0 process instances and the larger percentage of level 4 process instances in this set compared with the data relating to all the assessments shown in Figure 17.

Clearly many other such analyses are possible. Further analysis of this type are planned.

### Conclusions About Assessments and Ratings

The major findings from the analysis presented in this paper are:

1. Only two regions have participated in the trials by providing data thus far (i.e., December 1997): Europe and South Asia Pacific2.

---

2 Since we performed this analysis, data has been collected from Canada and Latin America, USA, and the Northern Asia Pacific.

---

**Software Process Newsletter: SPN - 15**
2 We have data from 30 assessments conducted in 23 different organisations in these two regions.

3 There was a good distribution in terms of Organisational Unit size (both large and small). However, there was no participation for OUs whose primary business sector was: business services, petroleum, automotive, aerospace, public administration, consumer goods, retail, health and pharmaceuticals, leisure and tourism, manufacturing, construction, and travel.

4 Most assessments involved only one project in the OU.

5 All processes in the version of the ISO/IEC 15504 Reference Model that was evaluated were covered. The median number of process instances per assessment is seven.

6 In general, we found that the attributes corresponding to the higher capability levels less often receive the higher ratings than those corresponding to the lower capability levels.

7 In a significant number of cases, process instances fail to achieve a particular capability level because of inadequacies at the previous level, rather than at the level in question.

8 In approximately 19% of the process instances were at level 0, 50% at level 1, and 19% at level 2.

9 Approximately 19% of the process instances were at level 0, 50% at level 1, and 19% at level 2.

Acknowledgements

The earlier work of Ian Woodman [2] is acknowledged as is the assistance of Khaled El Emam for providing the box and whisker plot in Figure 14 and to John Wilson for discussions on the database issues involved in the above analyses.

References


Robin Hunter can be reached at: Department of Computer Science, University of Strathclyde, Richmond Street, GLASGOW G1 1XH, UK; Email: rbh@cs.strath.ac.uk
Figure 16: Distribution of attribute ratings by level.

Software Process Newsletter: SPN - 17
Figure 16: Distribution of attribute ratings by level (contd.).

process capability levels

Figure 17: Distribution of capability levels across all process instances.

Figure 18: Profile of process capability across all process instances per process category.

Figure 19: Process capability levels for high criticality products.
SOFTWARE PROCESS IMPROVEMENT IN CENTRAL AND EASTERN EUROPE

Miklós Biró, MTA SZTAKI, Hungary (moderator)
J. Gorski, Technical University of Gdansk, Poland
Yu. G. Stoyan, A.F. Loyko, M.V. Novozhilova, National Academy of Sciences of Ukraine
I. Socol, D. Bichir, SIVECO, Romania (INSPIRE INCO-Copernicus Project)
R. Vajde Horvat, I. Rozman, J. Györkös, University of Maribor, Slovenia

The worldwide information technology (IT) market has been growing at a rate of 8-10% since 1994. This growing market offers new opportunities for acquiring or increasing market share. A competitive software industry can be a cornerstone of economical growth in Central and Eastern European (CEE) countries if they can take advantage of this opportunity, exploit their traditional strengths, and overcome their weaknesses, all of which are briefly analysed below.

SWOT (Strengths, Weaknesses, Opportunities, and Threats) Analysis from the Perspectives of the Four Possible Levers of a Firm

Levers are the means used by a firm to multiply its resources. Fundamentally, it is the use of levers that account for the differences in profitability among firms. Four possible levers of a firm are the financial lever, the operating lever, the marketing lever, and the production lever.

What are the leverages that can be exploited by CEE companies or companies outsourcing their software development activity to CEE?

CEE has a number of general strengths including a highly-educated workforce that is able to assimilate new skills rapidly and produce high quality goods for export at relatively low costs. For the same reasons, R&D capacity is high as well.

Operating leverage is the relative change in profit induced by a relative change in volume. Because of its low operating costs, the CEE software industry has a high operating leverage. Consequently it can generate more profit than its less-leveraged competitors as soon as its volume reaches a given level.

The relative lack of local managerial skills and experience, and the former neglect of the development of a quality culture, are weaknesses that have an impact on both the production and marketing leverages.

Production leverage is the rate of growth of profits resulting from cost declines as a result of progress on the experience curve. Production leverage can be achieved only if management is able to properly organise production. Quality management is an important part in this organisation.

The two main ingredients of marketing leverage are higher prices and innovative distribution. The achievement of either of these goals requires highly-perceived quality and advanced market management skills. As far as production and marketing leverages are concerned, CEE is making efforts in training managers to obtain the necessary skills that were previously unheard of in the former economic system.

The possibility of making use of financial leverage—having and exploiting debt capacity—depends on the advent of general economic recovery and lower inflation, both of which require a rather long-term process.

The notion of leverage used in this section is expounded in more detail in [14].

Quality Awareness in CEE

Hungary

The general Hungarian quality scene is best characterised by the increasing number of ISO 9000-certified companies that have grown from very few at the beginning of the 1990's to over 500 today. However, up to now, few software development organisations have achieved ISO 9000 certification. One of these certified companies includes the Informatics and Systems and Control departments of MTA SZTAKI.

Regarding the capability maturity of software development firms, we assessed some software companies with the help of the BOOTSTRAP software process assessment methodology. According to our assessments, the maturity levels of assessed software-producing units were between 1.25 and 2.75.

To obtain a broader picture of quality awareness within the Hungarian software industry, we created a short questionnaire to which companies were asked to reply voluntarily and anonymously. Eighty-eight percent of respondents knew about ISO 9000 standards and 38% knew the BOOTSTRAP methodology. A few have heard about CMM, SPICE, and TickIT methodologies and standards, while other methodologies were not well known. The demand or requirement for formal certification has not yet become obvious. The majority of respondents (88%) do not or rarely require formal certification to ISO 9000 from their subcontractors. Neither are they usually required to have formal certification as a subcontractor. At the same time, the majority of respondents feel the need for formal certification of their quality management system. Some are planning for it or are currently undergoing certification. The initiations of quality management systems are present almost everywhere.

The second half of the questionnaire was directed toward specific areas of quality management. Questions were asked about the level to which processes of a specific area have accomplished or the existence and level of detail of certain documents, the answers of which could be chosen from a range of four levels. Results were, of course, not precise enough to be conclusive at a general maturity level, but were satisfactory enough to make comparisons about the awareness of the various quality areas.

Poland

During March-April 1997, a market study was conducted to identifying the need for improvement of software processes in Polish institutions. The study concentrated on large institutions, based on the assumption that small and mid-sized enterprises do not have enough capital to invest in technology improvements (in Poland there are no well-defined schemes to support small and mid-sized business in technology advances). Sixty mostly-large institutions were contacted. They have been divided into three categories:

1. suppliers of IT infrastructure (hardware and software) and system integrators for end-users (for example, HP, Oracle, Unisys, and Computerland)
The results show that there is interest in software technology transfer and in process improvement. An interesting observation is that client organisations seem to be more aware of this need than, for example, software development companies. It can be concluded that there is a very strong feeling that something must be changed. However, this feeling has not been directly translated into a deeper understanding of what should be changed and where the investments should be directed. Ultimately, the market needs more awareness-building activities, more success stories, and more demonstrations of positive examples.

Almost all institutions that responded positively declared an interest in courses and training. Again this finding shows the need for awareness-building activities. It also provides a chance that such training seminars can lead to more concrete cooperation in the future.

Romania

More than 90% of Romanian software companies are private. State-owned companies still exist (no more than 10 in the whole country) such as the Institute of Research in Informatics. To raise the quality of software, ISO 9000 certification has been strongly emphasised as important. To stimulate the ISO 9000 certification initiatives, the ISO 9000-certified software companies will be provided with many incentives, such as reduced or no taxes.

Software Technological Parks will be created and/or implemented and new standards will be developed for software engineering (such as mandatory and optional standards, recommendations, a Romanian keyboard, and Romanian IT terminology). Public administration activities will be fully informatised to improve their services and to simplify their procedural and administrative practices.

Training is another important problem. It is being provided to change the structure of the specialised faculties and to add new domains, such as project management, marketing, and quality assurance for the IT industry. Strong cooperative efforts will be established between education and the software industry by giving students the opportunity to achieve practical experience by working in the software industry. The student’s scholarship will be tax-free.

Many of the unemployed will be absorbed by a collateral industry, data production. A large number of software companies will be involved in this activity and it is estimated that about US$20 million will be earned from exporting data production.

Slovenia

At the beginning of 1994, the Laboratory for Informatics at the University of Maribor, the Ministry of Science and Technology of Slovenia, and the Slovenian local industry (11 organisations) initiated a project called PROCESSUS—Assessment and introduction of quality systems. Two main goals were defined for the project:

1. Research: development of a methodology that could be applied for SPI within a wide range of software companies and that would comply with ISO 9001 and CMM. [6, 7, 8, 9]

2. Implementation: use of the methodology to introduce and maintain quality systems in participating organisations.

The first iteration of the project has finished and the results of the PROCESSUS project prove that the developed methodology is being directed toward the right goals [10, 11]. Statistics of achieved results within cooperating organisations are presented below.

- **Large information organisations (1).** The primary goal of this organisation was to use the methodology for consultation in activities with other companies. Some projects are already being launched in different software companies using the PROCESSUS methodology.

- **IT departments within large enterprises (3).** One of these IT departments has already arranged all of its procedures in accordance with software quality system requirements. The enterprise has already achieved the ISO 9001 certification.

- **Independent software companies (7).** Two independent companies have already achieved certification. Another two companies have cooperated, with the intention of applying for ISO 9000 certification. Within these two organisations some of their procedures have improved.

- **Software clients (sometimes with a large software department), for example, banks, insurance companies, administration.**

- **IT departments within large enterprises (3).** One of these IT departments has already arranged all of its procedures in accordance with software quality system requirements. The enterprise has already achieved the ISO 9001 certification.

The next iteration of the project is being initiated and will be based on the developed model, experiences with partners from the first iteration of the project (the activities for personnel motivation will be emphasised), and market interests. Market research has shown that 9% of independent software companies have already achieved ISO 9000 certification. Twenty-five percent of organisations have already started an SPI project and 30% of organisations intend to start an SPI project.

Ukraine

In Ukraine, the state system of certification of production (so named UkrSEPRO) has been working since 1993. Ukrainian state standards of quality (SSU) started in 1996 with an immediate application of ISO 9000.

In 1996, within the framework of cooperation with European organisations [12][13], the Ukrainian experts have been taking part in the project Qualification of Ukrainian Software Specialists (QUALUS) maintained by the German government’s TRANSFORM program. An international seminar entitled "Quality of software: The information market" had been organised in which the report of Jurgen Heene, director of WIDIS GmbH, entitled “German experience in certification on ISO 9000” was discussed. In Kiev, the expertise of WIDIS GmbH was used in a number of consultations on the installation of quality systems.

The analysis of activities within Ukrainian firms engaged in the IT field shows that the introduction of quality systems depends on the following major factors.

- **The requirements of the customer.** There are essential differences between the IT market within Ukraine and outside of it. The lack of quality systems and information about its features prevents the progress of production of Ukrainian firms in external markets. Moreover, many Ukrainian software firms working on projects for foreign customers have lost contracts, with resulting financial losses, as a result of
the absence of the necessary technology to maintain the quality of production. Within the Ukrainian market, only the government has required certification and the introduction of a quality system as part of the bidding process of tender participants. Sometimes those enterprises that need IT within the manufacturing production process for export to their foreign customers require its certification. The authors of this paper have seen that fact at Kharkov turbine factory when they were selling their own CAE system. The need for certification of the factory’s production was because the customer was foreign.

- **Scale of manufacturing.** It is obvious that when small enterprises are engaged in simple and non-labour consuming production, the problem of introducing quality systems and certifying their production processes is, as a rule, as yet unrealized. With an increase in the scale of manufacturing, the problem of how effective the organization can be in taking measures to maintain levels of demanded quality becomes more urgent.

- **Qualification of the employees of the firm.** For an organization to be effective in its manufacturing processes and to meet the level of quality that its customers demand depends on the knowledge and qualification of both managers and employees of the firm. In Ukrainian software firms working on large projects, the emphasis is on the use of various organisational technologies and on group work: drawing up the technical projects, organisation of working meetings, independent testing, and so on. Thus, the questions of quality are not allocated to a separate problem, but are considered to be necessary results.

### Conclusions

We claim that it is possible to increase competitiveness in CEE and in countries outsourcing their software development activity to CEE simultaneously by way of mutually-fruitful cooperations, the precondition of which is the assessment and improvement of the capability of the CEE software industry. This joint interest manifests itself by several European Commission-supported initiatives, including the ColorPIE ESBNET ESSI proposal, the PASS ESSI PIE project, the INSPIRE INCO-Copernicus project, and so on.

We would like to draw attention to a fact that must be taken into account if we want to achieve real results. In CEE countries, the IT business is conducted mainly by SMEs that are too small to invest in SPI. Usually they do not have enough capital and are very much opportunity driven. It is apparent that there is a need to further develop SPI models that would be applicable to SMEs, not only in CEE but in other parts of the world as well.

Most of the present models target large companies and are too "heavy" to be applicable to SMEs. The main difference should be that the feedback loop from the investment to the actual benefit should be much shorter and the investment should be split into small slices. Without having such a model it is rather unlikely that SMEs will be able to enter the improvement path in a planned and systematic way.

### References


Miklos Bíró can be reached at: MTA SZTAKI, Kende u. 13-17, Budapest, H-1111 Hungary, E-mail: miklos.biro@sztaki.hu

### EUROPEAN SPI-GLASS

Colin Tully

Colin Tully Associates

**SPI - different paths to salvation: continued**

In the last issue, we embarked on a comparison between ESSI (the European Systems and Software Initiative, part of the European Commission’s ESPRIT programme) and the SEI’s software process programme, as contrasting mechanisms for driving the adoption of SPI (software process improvement).

Two contrasts were drawn. The first contrast, with respect to theoretical basis, was between the software management theory embodied in the CMM, on which the SEI’s process programme is based, and the economic theory underpinning ESSI. The second contrast, with respect to model adoption, was between the USA’s monotheistic faith in the CMM and the combination of polytheism and atheism found in Europe.

Perhaps crudely, the difference between the two approaches, as characterised so far, may be summed up as “give them the tool” (USA) and “give them the money and let them decide on a tool” (Europe). Some readers may find themselves recalling arguments over third-world aid, in which a similar dichotomy occurs. It should be
stated at once that this column takes no position on the relative merit of the two approaches.

We now continue our comparison of the European and American approaches by considering three more contrasts.

**Third contrast: funding channels**

Saying that the two programmes are dominated by different "theories", one of which is economic, should not disguise the fact that of course both programmes have an economic dimension. In both cases, substantial sums of public funding have been deployed, reflecting the concerns felt by European and American administrations about the strategic importance of software capability.

It is not our purpose, even if the data were available, to compare the amounts of public money invested in these initiatives. It is more interesting to compare the ways in which funds are channelled. The contrast is between indirect funding for the promotion of SPI in the States and direct funding of specific SPI projects in Europe.

**Indirect funding of improvement in the USA**

Funding from the US Government (specifically the Department of Defense) to encourage SPI, has, however, been indirect. It has been channelled in two main ways.

First, by funding of the SEI in general, and in particular through continued budgetary approval for the SEI's software process programme over many years, the DoD subsidised the development of the CMM. Without that investment, the main engine for the spread of SPI in America would not have existed.

Second, by exploiting its massive procurement power, and by mandating specific maturity levels for procurement contracts, the DoD fostered the first (critical) phase of industrial take-up. Without that commercial incentive, the CMM might have remained of mainly academic interest.

The investment and the commercial incentive were, of course, aimed just at the defence sector of US industry, not explicitly at American software producers as a whole. The DoD has demonstrated some uncertainty over whether its sponsorship of the SEI should extend to supporting widespread roll-out of the CMM across all industrial sectors. Nevertheless, the SEI has taken what chances it could to promote such roll-out; and that push, combined with the pull of US industry's cultural propensity to accept new ideas, has ensured the take-up of CMM-driven SPI in thousands of software-producing organisations.

It is interesting to note that there has been an attempt, conscious or otherwise, to replicate some elements of this approach in Europe. The attempt was quite unrelated to the ESSI programme which is the subject of this column's current attention. The Commission funded the development of a model called Euromethod, to encourage the customer-contractor process in the public-sector procurement of large information systems. That investment was then to be followed by the commercial incentive of mandated use of Euromethod in bidding for such contracts. It is not clear what degree of success Euromethod has had in meeting its initial goals; but it seems so far not to have achieved the second-phase roll-out that characterised the CMM's success in America.

This funding can be described as indirect because in no case did the US Government directly fund specific projects for SPI. In the first stage it established the SEI, and allowed it to exercise its own judgment about how to deploy the funding it was given. In the second stage it simply said to contractors, "If you want to be allowed to bid, show us you've improved to a specified level" - a requirement that struck immediately at a substantial proportion of contractors' business. There was no money devoted directly and specifically to develop a model or to meet the costs of improvement.

**Direct funding of improvement in Europe**

Funding from the European Commission has, by contrast, always been directly channelled to specific projects, undertaken by industrial consortia with or without academic input. Bids for project funding have been required in all cases to comply with Commission work programmes in force at the time, so that the Commission has exercised fairly tight control over the nature of project proposals.

With respect to SPI, projects have been of two kinds. First, as part of mainstream R&D within the Commission's long-running ESPRIT programme, there have been a number of projects to develop models and methods. Leading examples include Bootstrap (a process maturity appraisal model), ami (a method for introducing metrics into the software process), and REBOOT (a method and appraisal model for introducing systematic reuse into the software process).

These projects are generally recognised as having produced models and methods of good quality. Effort has been put into publishing and disseminating their results, and into establishing various mechanisms to try to promote take-up after the projects have ended: the Bootstrap Institute, the ami User Group and a number of REBOOT follow-on projects.

Inevitably, however, such individual take-up mechanisms cannot match the impact of the SEI, which represents a massive statement of long-term government commitment. The problem with a project is that by definition it is of limited duration. However excellent its results, and however strong the commitment of its participants, momentum is almost bound to fall away at the end of the project - at the very point where, if take-up is to be achieved, momentum needs to increase.

Further, the project-oriented approach by its nature produces a multiplicity of models and methods, with no provision for their integration. It thus directly leads to the polythetic fragmentation described in the last issue.

Projects of the second kind have been best practice projects - constituting the ESSI component of the ESPRIT programme (in other words, support the direct marginal costs of SPI projects within individual software producing organisations. As we have already observed, this is a radical shift from the technological innovation sought by normal ESPRIT projects (the development of new models and methods) to organisational innovation (the development of new practice).

Nevertheless, support is still for a single limited-life project. Proposals for support are required to demonstrate the organisation's commitment to longer-term SPI. But, even if that commitment is honestly made at proposal time, it may prove hard to sustain two or three years later, after the end of the project. In the end, funding is offered and accepted for a tightly bounded package of work, not for a long-term programme. The contrast with the sustained effort needed to achieve level 3, to qualify for the DoD approved list, is stark.

**Fourth contrast: exercise of central authority**

This comparison follows directly from the nature of the funding channels, as just discussed. It can be presented briefly.
Controlling the model, and controlling capability, in the USA

The exercise of central authority in America has been shared between the SEI and the DoD, each with a clear role.

The SEI exercises central authority to control the CMM, its development and deployment. The major parts of its role are:
- to be the development authority for the CMM and CMM-based methods;
- to promote, and to exercise quality control over, their application;
- to maintain a data repository of appraisal results; and
- to publish and disseminate information.

The DoD exercises central authority by setting required capability levels for its approved contractors, and by conducting capability evaluations in selecting for specific contracts.

Controlling projects, in Europe

Central authority is exercised by the European Commission over the projects it funds, in three ways:
- defining the broad parameters of projects from time to time, in terms of subject matter, consortium structure, eligible costs, funding limits etc;
- evaluating project proposals, to select the specific projects to which funding is to be awarded; and
- exercising quality control over projects selected for funding, chiefly through detailed contract negotiations before the project, and through review of key deliverables during and after the project.

Some readers may reflect that there is some similarity between the DoD’s role and the Commission’s role, insofar as they both enter into projects as one side in a customer-contractor relationship. The differences are substantial, however. In the DoD’s case, it funds projects with the intention of acquiring real delivered systems, of which it will be the user. In the Commission’s case, it funds projects with the intention of enhancing industrial capability, of which the “users” will be not the Commission itself but the project participants.

Fifth contrast: improvement drivers

An empirically based comparison can be drawn between the predominant drivers for SPI in American and European organisations. Again, this contrast can be presented briefly.

Model-driven improvement in the USA

By far the predominant driver for American organisations that have embarked on SPI is to climb the scale of maturity levels, as a result of undertaking CMM-based appraisals. This is a natural effect of the dominant position of the CMM in the States. Such companies may be described as model-driven.

Being model-driven determines the priorities for improvement, depending on an organisation’s current level. At level 1, the prescribed focus is on the set of level 2 key process areas; and similarly for level 2 and above.

There are exceptions to the model-driven norm. A small proportion of companies, including acknowledged SPI leaders such as Boeing, Hughes or Motorola, have carried out in-depth analyses of the business importance of the software process, from which they have developed company-specific improvement priorities and programmes. Within those strategic programmes, the CMM always plays a role, providing a key performance indicator and a default set of improvement priorities. But it is a part in a much larger whole: the driver is process improvement for its own sake, and such companies can be said to be process-driven.

Single-issue-driven improvement in Europe

The predominant approach in Europe is to identify a single process issue on which to launch a SPI initiative. This is a natural effect of the diversity of models and methods, and of the short time-scale of ESSI projects. Such companies may be described as single-issue-driven.

Known examples of such single issues include the introduction of object-oriented technology, client-server architecture, metrics, project management, requirements capture, reuse, testing, and defect management. These are all process changes, but many of them are focused on the methods and tools that support various key process areas (to that extent they have a strong technological flavour) rather than on the process seen as a whole or on process assessment.

Europe also has SPI leaders, such as Ericsson, Philips and Siemens, who have graduated to being process-driven in the same way as the American leaders discussed above.

To be concluded

In the next issue, European SPI-Glass will conclude its comparison of features of American and European SPI.

Acknowledgement

Some material in this article is adapted from a forthcoming book to be published by John Wiley & Sons Ltd.

Colin Tully can be reached at: Colin Tully Associates, 97 Par Meadow, Hatfield, Hertfordshire, AL9 5HE, UK. E-mail: c.tully@herts.ac.uk.
The SPIRE project aims to lower the barriers to successful software process improvement by Small Software Developers (SSDs), defined as organisations employing up to 80 software staff, including small software companies and small software units in larger organisations.

The SPIRE project is assisting over 60 SSDs in four European regions (Sweden, Italy, Ireland and Austria) to carry out short mentor assisted software process improvement projects. Experienced mentors guide the SSD through an assessment of needs, the preparation and assistance in developing a detailed plan for a cost-effective small software process improvement project, implementation of the project and evaluation of results.

The assessment of needs entails the mentor working with the SSD to define the organisation’s business needs and assist in setting up a SPICE self-assessment model using one of two software tools (Bootcheck or Synquest) which embody a SPICE version 2 compatible assessment model and which provide assessment results as SPICE conformant profiles. These tools are ideal for use in mentor assisted self-assessments which are completed within 3 to 5 hours. Additionally, a separate confidential staff attitudes survey is undertaken. The whole process is completed in a one day on-site visit.

Following analysis of the results, priority areas for process improvement matched to business needs are defined which provide the basis for discussion of potential improvements. The SSD then proposes a focussed improvement project which must be completed within 6 months and must demonstrate quantifiable business benefits. The improvement project plans are then submitted to a regional panel which review and approve the individual projects. The SSD can obtain support funding of up to 110K ECUs as a contribution to its own costs plus it also has the assistance of an experienced mentor for up to 10 days free of charge. The mentor will ensure that the project maintains momentum to completion.

At the end of the improvement project, a second mentor assisted self-assessment is performed to compare before and after process capability results. It is intended to submit some of the SPIRE assessment results to the SPICE trials phase 2. This will provide valuable data for comparison of assessment approaches and also quantitative data following completed process improvement actions.

Based on the experience gained in these projects, SPIRE will generate case studies and other deliverables of value to all SSDs, and disseminate them widely throughout Europe. The experiences gained are expected to have a major impact on company awareness of the benefits of software process improvement.

The SPIRE project started in March 1997 and runs until September 1998. All the initial assessments have been completed, with improvement projects being performed between March and August 1998.

The SPIRE consortium consists of the Centre for Software Engineering (Ireland), Etnoteam (Italy), IVF Centre for Information Technology (Sweden), ARC Seibersdorf (Austria), SIF (Northern Ireland). SPIRE maintains web sites at all partner sites. The home page can be found at <http://www.cse.dcu.ie>.

**PULSE**

The PULSE project is one of 9 projects funded under the EC’s new SPRITE pilot programmes. The SPRITE programme aims at the application, validation and demonstration of existing and new instruments of support and guidance of software and systems procurement. All projects are linked to standardisation initiatives. All projects commenced in January 1998 and will run for 12 months.

The PULSE project aims to combine two approaches for assisting organisations to improve their procurement processes; defining and verifying a formal methodology for identifying and assessing the processes used by an organisation for IT procurement, and identifying a set of organisational actions that improve the way in which procurements are managed and the success of IT procurement teams.

The PULSE project will achieve its aims by: developing a methodology with associated tools to allow organisations to assess and benchmark their procurement capabilities and to determine those areas where improvement actions should be taken in order to meet their specific business objectives, and identifying new organisational and communication techniques that allow better integration and teamwork between the three key areas (purchasing, technology development and strategic planning and standards) for any IT procurement. The parts of the project are known as the PULSE methodology and TEAM working aspects respectively.

As part of the PULSE methodology the project will:

- develop an acquisition process reference model
- develop a detailed acquisition assessment model
- define an appropriate assessment method
- develop a software based assessment tool
- trial the assessment method with user partners across Europe
- define a training syllabus and certification scheme for assessors
- develop a methodology licensing scheme
- present the PULSE reference model to ISO as a plug-in extension to an existing standard

The original scope of SPICE was intended to include assessment of the customer acquisition processes. It was recognised that project success depended on the capability of the acquisition partner as well as the supplier.

The development of SPICE was predominantly undertaken by the world’s experts from the software engineering community and from the influence of major purchasers wishing to assess the capability of their software suppliers. The original intent of assessing customer acquisition processes somehow sidestepped and customer-supplier processes provided the main focus in the model.

The PULSE project will ensure that the intended focus is put back on track. The project has already researched a representative set of procurement practices and existing models around the world. By the end of March it will have developed an acquisition reference model as a plug-in extension to the ISO/IEC 15504 reference model. A detailed assessment model and software assessment tool will then be developed and assessments will be undertaken in major European to validate the model.

The PULSE project has already created significant interest from major players outside the project in
Australia, the UK and Hungary. A major milestone in the project will be the presentation of the PULSE reference model at the ISO meeting in South Africa in May this year. Based on the experience gained thus far in the development of the PULSE reference model, input is also being provided to the revision requirements for ISO/IEC 12207 Software Lifecycle Processes, which are being finalised in Venice in the Spring of 1998. Expectations are that ISO/IEC 12207 processes will be extended to include systems engineering and system acquisition processes.

The PULSE consortium consists of IVF Centre for Software Engineering (Sweden), ATB (Germany), CR2A-DI (France) and the Open Group (UK). The project also has 12 major associate user partner organisations represented from, amongst others, the defence, aerospace, pharmaceuticals, industrial and public administration sectors.

The PULSE project manager can be contacted at adg@ivf.se

Alec Dorling can be reached at: SPICE Spotlight, IEEE Software Process Newsletter, IVF, Centre for Software Engineering, Argongatan 30, S-431 53 Mölndal, Sweden. Email: dorling@bcs.org.uk

ANNOUNCEMENTS


General Chair: Bashar Nuseibeh, Imperial College, London, UK

Programme Chair: Volker Gruhn, University of Dortmund, Germany

Programme Committee:
- Nacer Boudjlida, CRIN, Nancy, France
- Jean-Claude Demrieme, CRIN, Nancy, France
- Gregor Engels, University of Paderborn, Germany

SOFTWARE PROCESS - IMPROVEMENT AND PRACTICE

Software Process - Improvement and Practice. Articles scheduled to appear in the next issue of the Software Process - Improvement and Practice journal, published by Wiley (http://www.wiley.co.uk), include:
- Evan Aby Larson and Karlheinz Kautz: “Quality Assurance and Software Process Improvement in Norway”
- Ashok Dandekar, Dewanye E. Perry, and Lawrence G. Votta: “Studies in Process Simplification”
- Jim Arlow, Sergio Bandinelli, Wolfgang Emmerich and Luigi Lavazza: “A Fine-grained Process Modelling Experiment at British Airways”
- Martin Verlage: “Experience With Software Process Modelling”

9th International Symposium on Software Reliability Engineering - CFP: This will be held on 04-07 November, 1998, in Paderborn.
EMPIRICAL SOFTWARE ENGINEERING
An International Journal

Editorial Board List: January 15, 1998

Editors-in-Chief:
Victor R. Basili, University of Maryland, USA.
basili@cs.umd.edu
Warren Harrison, Portland State University, USA.
warren@cs.pdx.edu

Associate Editors
H. Dieter Rombach, University of Kaiserslautern, Germany.
rombach@informatik.uni-kl.de
Ross Jeffery, University of New South Wales, Australia.
r.jeffery@unsw.edu.au
Koji Torii, Nara Institute of Science and Technology, Japan.
torii@is.aist-nara.ac.jp

Editorial Board
William Agresti, MITRE Corporation, USA.
wagresti@nsf.gov
Motoei Azuma, Waseda University, Japan.
azuma@azuma.mgmt.waseda.ac.jp
Lionel C. Briand, Fraunhofer Inst. For Experimental Software Eng., Germany.
briand@iese.fhg.de
Bill Curtis, TeraQuest Metrics, Inc., USA.
curtis@acm.org
Michael K. Daskalantonakis, Motorola, Inc., USA.
dask@mot.com
Michael Deutsch, Hughes Network Systems, USA
mddeutsch@hns.com
Norman Fenton, City University, London, UK.
n.fenton@csr.city.ac.uk
Robert Grady, Hewlett-Packard, USA.
watts@sei.cmu.edu
Chris Keminer, University of Pittsburgh, USA.
ckeminer@katz.business.pitt.edu
Frank McGarry, Computer Sciences Corp., USA.
frank_mcgarry.ssd1@cscmail.csc.com
Stan Rifkin, Master Systems, Inc., USA.
sr@seas.gwu.edu
Norman F. Schneidewind, Naval Postgraduate School, USA.
nbschneid@nps.navy.mil
Walter F. Tichy, University of Karlsruhe, Germany.
tichy@ira.uka.de
June Verner, Drexel University, USA.
June. verner@cis.drexel.edu
Anneliese von Mayrhauser, Colorado State University, USA.
avm@handel.cs.colostate.edu
Larry G. Votta, Bell Labs Innovations, Lucent Technologies, USA.
votta@research.att.com
Elaine Weyuker, AT&T Bell Laboratories - Research, USA.
weyuker@research.att.com
Marvin Zelkowitz, University of Maryland, USA.
mvz@cs.umd.edu
Stuart H. Zweben, Ohio State University, USA.
zweben@cis.ohio-state.edu
EMPIRICAL SOFTWARE ENGINEERING
An International Journal

Table of Contents - Volumes 1 & 2

Volume 1, No. 1, 1996

Editorial - Warren Harrison and Victor R. Basili
Peer Reviewed Articles:
  • Function Point Sizing: Structure, Validity and Applicability - Ross Jeffery and John Stathis
  • The Impact of Software Evolution and Reuse on Software Quality - Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaiichelvan and Nishith Gool
  • Comparing Ada and FORTRAN Lines of Code: Some Experimental Results - Thomas P. Frazier, John W. Bailey, and Melissa L. Corso

Viewpoint:
  • On the Application of Measurement Theory in Software Engineering - Lionel Briand, Khaled El Emam, Sandro Morasca

Volume 1, No. 2, 1996

In this Issue - Warren Harrison

Editorial - Victor R. Basili
Peer Reviewed Articles:
  • Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software - John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood
  • The Empirical Investigation of Perspective-Based Reading - Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sorumgord, and Marvin V. Zelkowitz
  • Increasing Testing Productivity and Software Quality: A Comparison of Software Testing Methodologies Within NASA - Donald W. Sova and Carol Smids

Volume 1, No. 3, 1996

In this Issue - Warren Harrison and Victor R. Basili

Peer Reviewed Articles:
  • An Instrument for Measuring the Success of the Requirements Engineering Process in Information Systems Development - Khaled El Emam and Nazim H. Madhavji
  • Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques - Christopher M. Lott and H. Dieter Rombach
  • Estimating Test Effectiveness with Dynamic Complexity Measurement - John C. Munson and Gregory A. Hall

Volume 2, No. 1, 1997

In this Issue - Warren Harrison and Victor R. Basili


Peer Reviewed Articles:
  • Computer-Aided Systems Engineering Methodology Support and its Effect on the Output of Structured Analysis - David Jankowski
  • A Replicated Experiment to Assess Requirements Inspection Techniques - Pierfrancesco Fusaro, Filippo Lanubile, and Giuseppe Visaggio
  • Monitoring Smoothly Degrading Systems for Increased Dependability - Alberto Avritzer and Elaine J. Weyuker

Volume 2, No. 2, 1997

In this Issue - Warren Harrison and Victor R. Basili

Guest Editor's Introduction - Lionel Briand

  • Empirical Evaluation of Software Maintenance Technologies - Filippo Lanubile

  • Fundamental Laws and Assumptions of Software Maintenance - Adam A. Porter
  • The Practical Use of Empirical Studies for Maintenance Process Improvement - Jon D. Valett
  • Qualitative Analysis of a Requirements Change Process - Khaled El Emam and Dirk Hoeltje
  • Evaluating Impact Analysis - A Case Study - Mikael Lindvall
  • On Increasing Our Knowledge of Large-Scale Software Comprehension - Anneliese von Mayrhauser and A. Marie Vans
  • Applying QIP/GQM in a Maintenance Project - Sandro Morasca
  • Early Risk-Management by Identification of Fault-Prone Modules - Niclas Ohlsson, Ann Christin Eriksson and Mary Helander
  • Problems and Prospects in Quantifying Software Maintainability - Jarrett Rosenberg
  • Experience With Regression Test Selection - Gregg Rothermel and Mary Jean Haroldi
  • Lessons Learned from a Regression Testing Case Study - David Rosenblum and Elaine J. Weyuker
  • NASA Shuttle Software Maintenance Evolution - Norman E. Schneiderwinder
  • The Study of Software Maintenance Organizations and Processes - Carolyn B. Seaman and Victor R. Basili
  • Report from an Experiment: Impact of Documentation on Maintainability - Eirik Tryggeseth

Volume 2, No. 3, 1997

In this Issue - Warren Harrison and Victor R. Basili

Peer Reviewed Articles:
  • How Software Engineering Tools Organize Programmer Behavior During the Task of Data Encapsulation - Robert W. Bowdidge and William G. Griswold
  • A Controlled Experiment to Evaluate On-Line Process Guidance - Christopher M. Lott
  • An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents - Lionel C. Briand, Christian Bunse, John W. Daly and Christiane Differding

Correspondence:

Volume 2, No. 4, 1997

In this Issue - Warren Harrison and Victor R. Basili

Peer Reviewed Articles:
  • A Study of Strategies for Computerized Critiquing of Programmers - Barry G. Silverman and Toufic Mehzer

Viewpoint:
  • Meta-Analysis - A Silver Bullet - for Meta-Analysts - Andy Brooks

Workshop Report:
The available titles are:

- Quality Modeling based on Coupling Measures in a Commercial Object-Oriented System
- Benchmarking Kappa for Software Process Assessment Reliability Studies
- SPICE: An Empiricist’s Perspective
- Defining and Validating Measures for Object-Based High-Level Design
- Implementing concepts from the Personal Software Process in an Industrial Setting
- The Internal Consistency of the ISO/IEC PDTR 15504 Software Process Capability Scale
- A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems
- A Case Study in Productivity Benchmarking: Methods and Lessons Learned
- The Repeatability of Code Defect Classifications
- Studying the Effects of Code Inspection and Structural Testing on Software Quality
- A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method
- Automated Software Engineering Data Collection Activities via the World Wide Web: A Tool Development Strategy applied in the Area of Software Inspection
- Evaluating the Usefulness and the Ease of Use of a Web-based Inspection Data Collection Tool
- Cost Implications of Interrater Agreement for Software Process Assessments
- Success or Failure? Modeling the Likelihood of Software Process Improvement
- Investigating Reading Techniques for Framework Learning
- A Comparison of Tool-Based and Paper-Based Software Inspection

International Software Engineering Research Network (ISERN) Technical Reports for 1998 available. ISERN is a community that believes software engineering research needs to be performed in an experimental context. By doing this we will be able to observe and experiment with the technologies in use, understand their weaknesses and strengths, tailor the technologies for the goals and characteristics of particular projects and package them together with empirically gained experience to enhance their reuse potential in future projects. ISERN consists of a group of organizations around the world conducting, sharing, and promoting empirical research in software engineering. The Technical Reports of ISERN for 1998 are now available on the Web at: "http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html".

Send Articles for SPN to:
Khaled El Emam
Fraunhofer - Institute for Experimental Software Engineering
Sauerwies 6, D-67661 Kaiserslautern, Germany
elemam@iese.fhg.de

All articles that appear in the newsletter are reviewed.

STEERING COMMITTEE
The members of the Steering Committee of the Software Process Committee are:

- Jean-Normand Drouin (Canada)
- Alfonso Fuggetta (Italy)
- Katsuro Inoue (Japan)
- Marc I. Kellner (U.S.A.)
- Nazim H. Madhavji (Canada)
- H. Dieter Rombach (Germany)
- Terry Rout (Australia)
- Wilhelm Schaefer (Germany)
- Lawrence G. Votta Jr. (U.S.A.)