Human, Social and Organisational Influences on the Software Process

Ian Sommerville and Tom Rodden

Cooperative Systems Engineering Group

Technical Report : CSEG/2/1995

CSEG, Computing Department, Lancaster University, LANCASTER, LA1 4YR, UK.
Phone: +44-524-593041; Fax: +44-524-593608; E-Mail: cseg-info@comp.lancs.ac.uk

Human, Social and Organisational Influences on the Software Process

Ian Sommerville and Tom Rodden,
Computing Dept.,
Lancaster University,
Lancaster LA1 4YR

is@comp.lancs.ac.uk

Abstract

This paper discusses some human, social and organisational considerations which affect software processes and the introduction of software process technology. We discuss how to analyse software processes as human rather than technical processes, how process improvement through the introduction of process technology may be limited because of non-technical factors and how processes are influenced by national and organisational cultures. In each case, we suggest the implications of these human, social and organisational factors for software process researchers.

1. Introduction

The objective of research into the software process is to develop methods and techniques for understanding and evaluating processes and providing support for process activities. By developing better ways of communicating and evaluating processes, we can propose and support process improvements based on organisational objectives such as improved software quality, reduced time to delivery of software products, reduced process costs, etc.

The vast majority of research on the software process and process support has had a technical focus. Therefore, research in process representation has mostly been concerned with developing notations for process modelling, with the development of support environments which are 'process aware' to a greater or lesser extent and with process enactment. For simplicity, we refer to all of this work as 'software process technology'. Many researchers believe that the most effective way to improve software processes is to deploy this technology in automating, as far as possible, parts of the process.

It is fair to say that there has been a relatively slow diffusion of software process research into industrial practice. Part of the reason for this is that technology transfer is always difficult. However, this paper argues that a more fundamental reason for the lack of acceptance of process support technology might be the mechanistic nature of the analysis underlying software process technology. Much work in this area uses computational metaphors (objects, data-flows, etc.) to describe the process and abstracts human activities into 'agents'. This mechanistic approach perhaps reached its zenith in Osterweil's contention [1] that 'Software processes are software too' where he advocated that process could be represented as programs which could be executed by automated or human agents.

The software process research community has now moved away from this extreme position and recognises that software processes are most definitely *not* programs. They are executed by people and not computers. People do not behave deterministically.

They have a variety of constantly changing goals and objectives and are influenced by a huge number of subjective factors in their working environments. They are governed by organisations and cultures which have significant, although sometimes subtle, effects on processes.

The move away from a mechanistic process is paralleled by a movement within business and organisational thinking away from strict structures. Much of this rethinking of organisations has been motivated by growing disaffection with "Taylorist" thinking which characterised members of an organisation as a "processing units" within a larger body. More recent initiatives such as Business Process Re-engineering (BPR) have sought to consider how businesses can be re-organised to operate in line with more directed business processes. These approaches involve the identification of a set of processes through which a particular activity is done, with a view to optimising the arrangement of those processes. These methods are particularly suited to a world in which the work is already organised into a set of relatively well defined and stable processes, These can be examined for duplication, redundancy, overlaps, etc. Most examples of successful BPR are from well established 'paperwork' operations such as Ford's invoice processing procedures [2].

However, the extent to which process models can be generalised to domains where the work is not of this well structured kind is open to question. Much of the work on engineering projects is of a problem solving kind -at the outset of a project it is not understood in detail how the project is going to carry through its objectives. How the work is to be actually realised will often be worked out over the course of the project itself. These natural uncertainties are often compounded by organisational settings, budgetary pressures, work overload, skill shortages, etc. For example, even when a software project is of the kind that has been done many times before it may not be possible to staff the project with those who are familiar with such work. This rich relationship between plans and the social setting in which they are carried out is reflected by Suchman [3] when she says that:

> "plans are resources for situated action but do not in any strong sense determine its course"

The relationship between process models and those who realise them is complex and a process model may be used in a variety of ways by different members of an organisation. These different orientations to the plan and the use of the plan as a means of guiding action are fundamentally social in nature and process modelling needs to consider the social setting in which models exist.

This paper discusses some human, social and organisational considerations which affect software processes and the introduction of software process technology. Although Lehman [4] and Curtis [5] highlighted these as important several years ago, researchers in software processes technologies have paid relatively little attention to these problems. Given the scientific background of most people working in this area, this is understandable as it is less well-defined and much more subjective than other aspects of software process technology research which may be assessed from a technical perspective.

We have drawn on work on social and organisational factors which we have developed in two separate projects. One of these was explicitly concerned with the studies of the software process as a social process. We observed software processes in two large organisations developing different types of product. In one case, the product was a software-controlled mechanical/electrical system; in the other, it was a safety-critical avionics system.

Our other work in this area is considering the human and organisational issues which affect the dependability of software processes, particularly the requirements engineering process. We have analysed types of process error which result from human interaction and are currently proposing guidelines for process improvement to reduce the probability of errors being introduced into a systems product.

We do not think it useful to comment here on any specific software process technology. As software engineers, we recognise the value of this research. However, we are concerned that a lack of awareness of human and organisational issues will make it impossible to realise the potential benefits of this technology. We believe that we must develop a better understanding of these 'soft' problems if effective software process improvement is to be achieved. In this paper, we therefore address three questions which are relevant to the development of such an understanding:

1. What methods might be used to analyse software processes and the rationale for these processes? We are particularly concerned with understanding these processes from a human and social perspective.
2. What are the human and social barriers to the successful software process improvement using software process technology?
3. What organisational and cultural factors might influence the adoption of software process technology?

These questions are addressed in separate sections of the paper. In each of these sections, we include a sub-section suggesting the implications for software process research.

2. Process Analysis

A notable omission in the software process literature is an almost complete lack of papers which describe existing processes except in very abstract and general ways. Whether this is due to a lack of process studies or whether such studies have been carried out but not published we do not know. However, the lack of information on real processes makes it very difficult to assess whether or not current research on software process technology can be applied in practical software development projects.

Of course, a number of researchers would claim that it is extremely difficult to make studies of practical software processes as the available languages for expressing software process models are still incomplete. It is only when we have a workable process description language that we can denote and analyse software processes. There is some validity in this argument. However, we do have here a 'chicken and egg' situation. Unless we study processes in detail we cannot derive realistic process modelling language requirements; unless we have such a language, we cannot study processes in detail!

In our work, we have taken the point of view that the existing approaches to process modelling are too mechanistic for describing processes which are dominated by human activities. In particular, current approaches to process modelling do not allow complex interactions between process participants to be represented and analysed. If we use such a process model as a tool for process analysis, we lose this important information. We therefore turned to the social sciences to discover methods of process analysis which might be applicable to software process studies.

The approach which we adopted for the studies of the software process was based on an ethnographic investigation. Ethnography involves an observer spending an

extended period of time living in a society or working environment and making detailed observations of its practices. Subsequent analysis of these observations reveals information about the structure, organisation and practices which take place in that environment. The ethnographer is a neutral observer; he or she should take an unprejudiced view and should not make judgements as to the practices which are observed.

Ethnography is useful because is concerned with what actually happens rather than some notional definition of what should happen. Ethnographers have no interest in classifying activities according to some pre-defined framework. This is important because the notion that there is a fixed process or procedure for most tasks is an over-simplification. Although there may be a formal division of responsibilities in an organisation and an allocation of roles to individuals, the practical reality is that the actual work done and the way in which it is done is continually re-negotiated at a very detailed level by the participants themselves.

We have had previous successful experience in applying an ethnographic approach to a study of the process of air traffic control [6] and other studies of work in different situations have revealed subtle but critical process information [3, 7]. As a follow-up to this work, we carried out ethnographic studies of the software development process in two application domains and studies of the requirements and specification process in other organisations.

We discovered that there were three areas where ethnography was valuable in developing our understanding of the software process. Existing notations for process models would not allow this process information to be represented. These three areas were:

1. Explicit identification of *ad hoc* cooperation. During a software process, we found that there were a large number of cooperation activities which were unplanned. A good example of such an activity is expert consultation. The development team knew who had particular classes of expertise and, when a problem arose, called on an expert to help with the problem. The consultations were informal and there was usually no written record of the discussion. We discovered that these informal activities were critical in the processes which we observed. Without these activities, the team could not have met its delivery schedule.
2. Identification of individual process interpretations. We found that different participants in the process had their own individual models of the process and ways of tackling particular problems. In any domain where there is significant creative input this is to be expected. However, it does contradict the notion that there should be a single, accepted process model which should define how the process should be carried out.
3. Identification of organisational influences on the software process. We found that the way in which a process was organised reflected organisational priorities rather than technical needs. For example, in one of our studies, the process was changed to satisfy a metric which the organisation had decided was an appropriate quality metric. The engineers involved in this process disagreed with this metric and believed that it resulted in lower product quality. Curtis *et al.* [5] discovered similar organisational influences which, in practical terms, reduced process and product quality.

We also found that ethnography was a very effective method of discovering process rationale. The non-judgmental nature of ethnography where we did not try to

explain in advance the objective of process activities meant that ethnographers had to ask why particular activities were carried out. In many cases, this rationale was invaluable for understanding the process. It was also the case that the rationale for an activity was often organisational rather than technical. In the example quoted above, engineers modified their testing process so that their performance as measured by an organisationally-imposed metric would improve. The process rationale, therefore, was organisational rather than technical although technical arguments for the metric could be made. The extent to which organisational motives are made explicit is, of course, dependent on the culture of the organisation.

We did discover, however, that ethnography was not an efficient way of discovering the overall process structure. Our previous experience with ethnography [6] had been successful in discovering the process. This is also true of other studies [7] [3]. What is notable about these studies, however, is that they considered environments such as control rooms where the duration of the process was relatively short and limited in both physical and organisational scope. The software projects which we studied lasted more than a year and we did not have the opportunity of observing a complete development from initial specification through to product delivery.

In one of our process studies, the application was concerned with developing a safety-related system and a very detailed set of procedures (in essence, a process model) had been defined. In the other case, however, we had no overall view of the process. We found it impossible to construct such a view from the data collected by the ethnographer.

Ethnography must therefore be considered as a means of putting flesh on the bones of a process model produced using more abstract approaches. Examples of ways in which the information derived from ethnography might be used to augment process models include:

1. Annotation with process rationale. As far as we are aware, no current approaches to modelling include explicit facilities to capture such rationale. If rationale for a model is not available, there are higher risks in changing the process. This is particularly true when the rationale is derived from organisational rather than technical requirements. In this case, it may not be obvious to those who have not been involved in model creation.
2. Identification of stable and variable parts of the process. It is likely that in many organisations there are relatively stable process fragments (such as reviews) and other fragments which vary considerably depending on the type of system being developed and the developers. Identifying this variability is important if automated support is to be provided as it should, initially at least, be focused on the stable parts of processes.
3. Communication pathway identification. The formal models of communication which may be included in process modelling languages (e.g. roles, responsibilities, etc.) are inadequate to reflect the rich actuality of communication in an organisation. Models should also reflect these forms of communication as they are often critical to the success (or otherwise) of a process.

These examples illustrate how the information derived from process observations may be associated with a process model. However, ethnography is essentially opportunistic. You can never know in advance what factors in a culture are important. Therefore, process models should not constrain the types of information which may be associated with them.

2.1 Implications for software process research

The principal implications of our process studies are that approaches to process modelling must be flexible and must be able to accommodate process variability. When a process is examined in detail, we believe that there will always be different, equally valid, interpretations of that process. It should be possible to express this variability in notations for process description. We are currently exploring the notion of process viewpoints [8] as a means of expressing this variability.

Improved notations to express cooperation and communication between process participants are needed. Current approaches to this communication description are based on relatively simple notions which do not take into account critical factors such as the status of participants and the context of the communications. Although there is no way in which implicit communications could be modelled in such a language, we believe that an improved communication notation would be an important step forward in improving the completeness of process descriptions.

Ethnographic analysis was very useful for understanding human interactions in the process and for discovering process subtleties which would not normally be represented in process models. We recognise, however, that the prolonged investigations which we carried out are impractical in most cases. We believe, however, that it is possible to produce a set of observational guidelines which will shorten the time required for process information collection. This will allow process information to be collected and used as annotations to process models.

We also feel that there is a need for the publication of significant process descriptions which could be used as exemplars by the software process community. The principal exemplar at the moment is the ISPW6/7 example [9] which is an artificial example covering part of the process. The example is plausible but the process fragment addressed may be unrepresentative and does not take into account any organisational influences. The publication of more realistic examples would represent a more challenging test for work in process representation. Of course, we recognise the problems this poses as organisations may feel that real process descriptions may be commercially sensitive information.

Methods and notations for describing process models should evolve from their essentially computational framework to allow informal information about human and organisational issues to be associated with them. They should also explicitly highlight those parts of the process which are likely to change. This does not just mean providing easy-to-use model editing; the notations for process should explicitly denote stable and variable process fragments.

We believe that it is particularly important to augment models with structured descriptions of the rationale for these models. This will this make the models easier to understand. It will also provide a basis for arguing for process improvements and serve as a basis for assessing the organisational impacts of these improvements.

2. Process Improvement

We believe that a fundamental objective of software process research should be process improvement of some kind. These improvements may have the objective of reducing costs, reducing the time required to complete a product, improving quality, etc. These improvements may be accomplished by introducing some process support technology and/or by changing the process (re-engineering it) in some way.

Interestingly, a survey of researchers from the software process community [10] disagreed with this notion. The survey showed that they did not think that either process

measurement or process improvement should be the principal objective of their research into software process modelling. Rather, process guidance was deemed to be the priority objective by the majority of researchers which were surveyed.

We are concerned about this allocation of research priorities. Organisations are interested in process improvement not the abstract intellectual challenges of process modelling, process analysis or process enactment. Unless the real problems of process improvement are taken into account, software process research will not have a significant industrial impact.

Most of the effort in software process improvement has been driven by the SEI's Capability Maturity Model [11] and by the adoption of ISO 9000 process quality standards. Many organisations have modified their processes to reach a certain level of process maturity [12] or to be compliant with ISO 9000. We are not aware of any analysis of the human and organisational problems of making these process changes so we do not feel that we can comment from this perspective on the SEI work. However, we note that Humphrey, the principal architect of the SEI model, has written about the importance of respecting professional skills when introducing process improvement [13].

This drive for improvement is also manifest in a number of methods of organisational change including Business Process Re-engineering (BPR) [14], and Total Quality Management (TQM) [15]. These methods share common concepts of 'process', and implicit or explicit mechanisms for monitoring 'process'. However, they tend to ignore the contingencies, interruptions, and problems which arise as work is undertaken in practice. This is perhaps most notable in the case of service industries where interpersonal communication skills are a recognised and prized resource. As Clements [16] states in his discussion of the use of IT in business process improvement.

> "The required skills are often vital to office operation, but are inherently difficult if not impossible to formalize and they lack the authoritative status they would enjoy if associated with higher-ranked more formally qualified personnel."

Clements argues for the need to allow development methods that allow the people concerned to bring their experiences to bear in an early and effective manner. Clements suggests that this is consistent with the traditions of Participative Design [17] and offers this as a solution for the design of new computer systems. While not fully endorsing his support of Participative Design particularly within an industrial context we would agree with Clements that early involvement of participants in the design and implementation of process improvements is essential.

2.1 Introducing process technology

We are not aware of any studies of the attitudes of end-users to software process technology. Given the relative immaturity of that technology and its lack of industrial use, this is not surprising. This section is therefore necessarily based on extrapolation from related work. There have been several studies concerned with user attitudes to the introduction of information technology in organisations. There has been some work done on human and organisational factors and software development environment design [18] with the problems of introducing groupware into organisations [19] and with the introduction of workflow systems into organisations [20].

The majority of interesting software processes are group processes so process support technology must explicitly address the problem of providing group support. In this respect, process technology is distinct from most CASE tools which are aimed at

supporting individual users. Grudin [19]has studied the general problems of introducing groupware into organisations. Given the nature of process technology, it is reasonable to assume that all of these problems are applicable. The problems which he identified are listed below. We have paraphrased Grudin's expression of these problems to relate them to software process technology.

1. *Disparity of work and benefit* People are reluctant to use technology which forces them to do extra work with no direct benefit. For example, it is recognised that software processes are dynamic and a requirement of a process support system might be to record process changes. Unless there is some benefit to engineers in recording such changes, they are unlikely to do so.
2. *Critical mass* Unless enough people in an organisation are willing to use process technology, it may never reach a critical mass of users to be useful. For example, if a process management system records current process state, this will only be useful if all members of a team use the system.
3. *Disruption of social processes* Process technology may not be accepted by software development teams if it interferes with the complex and subtle social dynamics of the team. Humans naturally adapt to other team members using implicit social and organisational knowledge and an awareness of cultural norms. It will not be possible in the foreseeable future to incorporate such knowledge in process support technology.
4. *Exception handling* Most human processes are characterised by flexible exception handling. The way in which an exception is handled depends on the type of exception, the context where it occurred, the resources available to the handler, other responsibilities of the handler, etc. If process support technology cannot support this flexibility without significant user input to the exception management system, then it is likely to be rejected by its users.
5. *Unobtrusive accessibility* Some features of a process support system are likely to be used frequently, others rarely. If rarely-used features cannot be used without a significant learning overhead, they will never be used.
6. *Difficulty of evaluation* We do not, at the moment, have any reliable methods for evaluating process support technology. Process metrics can, of course, be collected but relationships between these metrics and support technology are almost impossible to validate. Thus, generalising and learning from other's experience of process technology is very difficult.
7. *Non-intuitive requirements* It is difficult for software engineers to have strong intuition about the real requirements for process support. This contrasts with CASE tools, for example, where the tool developers could easily understand the problems of the tool users.
8. *Acceptance management* We do not currently know how to support the introduction of process technology into an organisation and manage its acceptance by project teams. As Grudin points out, individual tools are relatively easy to promote because they do not require 100% acceptance by all potential users before they are useful. If process technology is rejected by even a small number of end-users then its usefulness is significantly diminished.

Many of Grudin's identified problems were also found in a study by Le Quesne [18] which looked at the introduction of software development environments into

several different organisations. He found that social and organisational factors were more significant than technical factors in determining the successful introduction of the technology. Users of the technology did not see real benefits (although managers did), the useability of the system was very important, and it was impossible to evaluate the systems objectively. Other conclusions of this survey are covered later in this section where we discuss- the importance of professionalism.

The general representation of work and the re-engineering of business process have become closely tied with the properties of information technology. This relationship is most evident in the case of workflow systems where as Abbot and Sarin [20] state:

> "Workflow software provides the infrastructure to design, execute, and manage business processes on a network"

In outlining a set of different experiences in the use of workflow to support a number of different processes Abbot and Sarin emphasise the importance of considering the nature of the process in line with the supporting technology. They found that simply automating an existing process with no attempt at process improvement was counter-productive. The problems with the existing process were exacerbated when end-users had to deal with electronic versions of badly designed forms.

The way in which workflow systems is installed was found to be a very significant success factor for that technology. Abbot and Sarin state:

> " mundane issues of installation and administration, which arise with any software product, become more visible and critical.."

In summarising their experience gained from extensive examination of the use of a range of existing workflow systems a number of significant issues emerged which are of major importance to process technology. These include:

1. *Integrating procedural and non-procedural work* Workflow systems give pre-eminence to procedural representations of work. Procedural representations reflect the structured aspects of a process. Non-procedural work is of equal importance in ensuring the process is completed and workflow systems need to combine these in a flexible manner. Abbot and Sarin suggest that appropriate tools for enactment and relating workflow models to users offers the best solution and explicitly warn against trying to add non-procedural work to workflow descriptions.
2. *Support for external activities and meetings* Meetings played a central role in co-ordinating most of the workflow processes examined by Abbot and Sarin yet limited support was provided for meetings in the workflow model or the supporting technology.
3. *Evolutionary process development* Processes need to be allowed to develop and to fit to the social setting in which they are placed. This is particularly true in the case where the details of how an eventual goal will be achieved are unknown. Existing examples of workflow systems and software highlight limited support for process change or evolution.

2.2 Respecting professional skills

Most of the successful reports of business process re-engineering have involved clerical tasks such as invoice processing. By cutting across hierarchical structures in an

organisation, paperwork and the number of people needed to process that paperwork was significantly reduced. However, it is not clear if a comparable approach can be applied to processes which are largely based on professional judgement.

Software engineers consider themselves to be professionals. Their work is specialised and requires significant training and expertise. Like other professional tasks, software development requires the application of knowledge which is not specific to the organisation applying that knowledge. In general, professionals feel that they ought to have some autonomy in planning and scheduling their work and deciding the most appropriate way to solve problems. They think that they should be trusted to apply their professional skills without close supervision. Professionals have transferable skills and are willing to change jobs if they feel that their position is threatened by organisational change.

The threat to this notion of professionalism was identified by Le Quesne as a significant reason why the introduction of a software development environment was resented by its users. They felt that the changes to normal working practice which were required for the successful use of this system reduced their control over the ordering and pace of their work. They felt that the process changes and associated technology were de-skilling and did not recognise the fact that they were responsible professionals.

In our process studies in an aerospace company which had a process improvement programme in place, we found very similar attitudes. The software engineers felt that the changes which were imposed were not real improvements but were management challenges to what they considered to be successful working practice.

In some cases, they felt that the changes were politically motivated. Once such change was the introduction of an object-oriented development method after the requirements for a system had been specified using a functionally-oriented approach. The judgement of the engineers was that the cost of translation from a functional to an object-oriented model was likely to exceed any benefits from the object-oriented technology. However, their objections were over-ruled by management which had been imported from the company's principal site where the transition to object-oriented development had already been made.

The major complaint which these engineers had was not the method itself but the fact that, as professionals, they were not involved in the process of method selection. The need for involvement is confirmed in Le Quesne's study which showed that the only successful use of the software development environment was in an organisation where the software engineers were participants in the process of defining the environment requirements.

2.3 Implications for software process research

Using the example of electronic mail, Grudin suggests that successful applications to support groups are flexible, have a low learning overhead and do not incorporate notions of role, process and social interaction. He suggests that groupware applications are more likely to be accepted if they evolve from currently user applications so that a large learning overhead is not required.

This notion is confirmed by the relatively limited use of large-scale software development environments. These involve a major transition and change of working practice. Organisations and individuals prefer incremental change so tools which support individual tasks and which can be used or not are inherently more acceptable.

As far as software process research is concerned, this poses a major challenge. There has been a great deal of work carried out in the development of process-centred

environments. Examples of such environments include IPSE2.5 [21], Arcadia [22], MARVEL [23], ALF [24] and OIKOS [25]. Some of the developers of process-centred environments have suggested that the reason for the limited use of current SDEs is the lack of process support. By including such support, the attractiveness of SDEs to customers will be increased. If the analogy between process technology and groupware is appropriate, this may be an over-optimistic viewpoint.

Grudin's observation that successful groupware systems have evolved from existing systems suggest that an appropriate starting point for developing process support might be existing CASE tools. This, of course, does not negate existing research on what process support facilities might be required. However, it does suggest that researchers with an interest in the practical exploitation of their work should also consider the integration problems with existing systems.

As we have discussed, software engineers consider themselves to be professionals. They consider that a distinguishing aspect of professionalism is that professionals have some control over their own work activities. They exercise judgement about the best way to tackle a task and strongly resent organisational imposition of particular work practices. They are much more likely to accept change if they participate in the design of that change.

Of course, organisations may impose a technology but people have an innate ability to 'work around' such impositions and derive alternative working practices which circumvent the system. As Hirscheim and Newman discuss [26], end-users who have unwanted technology imposed on them are ingenious in finding plausible reasons why the technology is inadequate and should be rejected.

Given that an organisation is willing to involve its engineers in process change planning, this requires that the process technology used must be adaptable. It should be presented in end-user terms and it should be easy to understand process models and descriptions. It should be possible to construct (or at least to adapt) models and support tools without extensive training.

From the experiences of workflow systems, it is clear that there is a need for process presentation facilities so that end-users can understand models which have been developed and can participate in their development. Facilities need to be developed to link meetings and other external activities into the process. Future systems should incorporate flexibility in decomposing processes, library and re-use facilities which allow existing processes to be changed to meet current needs.

3. Organisational and Cultural Factors

All interesting software processes take place within a broader organisation of some kind which is itself constrained by legal, political and economic considerations. The interests in the software process range from the very direct interests of the participants in that process through to the very indirect interests of society in general. This range of interests is sometimes expressed in what is called a socio-technical pyramid. The direct involvement with the process and process support increases towards the apex of the triangle.

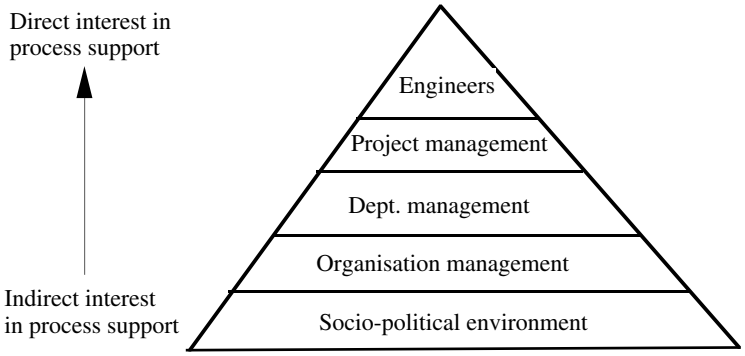

Figure 1 The socio-technical pyramid

Although there some of the process influences may be very indirect, this does not mean that they are insignificant. As an example of this, in UK industry, there is increasing pressure on companies to be certified for quality according to ISO standard 9000. This has now reached a stage where quality certification, irrespective of its relevance in a particular context, is expected of companies. In some cases, it is a requirement for sub-contractors. They must conform to this pressure in order to remain in business even when it is unlikely to make a significant difference to their product quality.

In this section, we consider three levels in this socio-technical pyramid. We firstly discuss the engineering level and the influence of systems engineering processes on the software process. We then look at the management level and the potential conflicts which can arise between technical goals, project management goals and business goals. Finally, we discuss the more indirect cultural level and the affect that cultural factors may have on the introduction of organisational change.

3.1 Systems Engineering

The vast majority of research on software processes seems to have considered software processes in isolation rather than in the context of some broader process framework. While it is understandable that research should be focused, a danger of such focusing is that critical considerations which derive from broader considerations are ignored.

In many organisations, software processes are only one of many engineering processes which are required to produce a product. These organisations have a general systems engineering process and this constrains other sub-processes which are part of it. To give a simple example, the systems engineering process may require the publication of interface specifications early in the process so that concurrent hardware development can proceed. The time required for tooling for manufacture is such that any delay in publishing these interfaces will result in high cost penalties and product delivery slippage.

This means that the software process used must deliver these specifications even if this results in a less than ideal process. An extended period of prototyping (say) may be desirable but may be impossible because of the need for early interface definition. Of course, defining the interface prematurely may cause problems later in the process. However, the costs of repairing these problems may be significantly less *for the organisation* (although they may mean higher software costs) than the costs of later interface delivery.

We were involved in an empirical study of the software process within a company which produced complex products which included mechanical, electrical, electronic and software components. The product which incorporated the software was being developed using a concurrent engineering approach with very tight delivery schedules. These schedules were very critical for the team concerned as failure to deliver on time could mean a transfer of future engineering work to another site.

The development process used meant that hardware from other products was modified and control software developed for this machine. We did not study the whole development process but examined the process during the development and integration of the software and hardware. Notable features of the process were:

1. There was no separate software test plan. The whole system was tested using standard procedures which had been developed for other similar products. The test sequence was not documented and the tests themselves were chosen by the testers depending on their knowledge of likely problems and previously identified system failures.
2. The engineering team met daily to assess problems identified during system testing to discuss how these problems may have arisen. Responsibilities for fixing the problems were then allocated. Because of the costs of modifying hardware, the software development team sometimes had to change their software to work around a hardware problem.
3. The process relied heavily on the use of experts who had detailed knowledge of parts of the system and their interfaces.
4. Apart from problem reports, very little documentation was maintained. The software evolved daily with no configuration management and no attempt to maintain links between problem reports and software system versions.

The company had, in fact, a detailed software process model which was simply ignored by the managers of the product development process. Their attitude was that the software process model may be applicable to software development projects but was quite inappropriate for product development to a very tight schedule.

In terms of the SEI model [11], this was undoubtedly a chaotic, Level 1 process. Yet in terms of product development, it was effective and the company concerned is a very successful multi-national organisation which relies on these types of product for most of its revenue. Systems engineering considerations meant that the conventional wisdom of the value of defined software processes simply did not apply.

Of course, this is a special case. We do not claim that this is a typical process and it is certainly nothing like the software process we observed for another systems product. However, it was equally true in that situation that the software process was determined by the systems engineering process. The lesson which we took from this is that to convert current software process research into practice, we must be very aware of these broader systems considerations.

3.2 Management and Organisational Goals

The people involved in an organisation generally have a range of different goals. These goals depend on their responsibilities and status in the organisation, their personal involvement with the organisation (e.g. they may own shares) and external circumstances such as family commitments. The goals of different people in an organisation are often opposing and an important role of management is to reconcile these opposing goals.

These differing goals sometimes arise because members of the organisation have different views of the organisation. If we take a University as an example, some professors may consider that the primary purpose of the organisation is teaching, others may see it as a research institution. University accountants see the organisation as a business enterprise. There is no single homogeneous view and we believe that this is true for all non-trivial organisations.

With this in mind, we can distinguish three related classes of goal:

1. Technical goals. These are concerned with the application and development of professional skills as discussed above.
2. Project management goals. These are concerned with achieving budget and schedule targets and ensuring the successful completion of projects.
3. Organisational management goals. These are concerned with ensuring the continued existence and development of the organisation as a whole.

Because of the pervasiveness of software, software process changes have a very significant effect in organisations and all of the above goals are influenced by software process change. At a technical level, the actual software development activities may be carried out in a different sequence, different methods may be used or new support tools introduced. At the project management level, managers must assess the costs and benefits of the change and find a way of introducing that change without adversely affecting project planning. At the organisational level, software is now so important that software failures can threaten the existence of the entire organisation.

Technical, project management and organisational goals do not, of course, all have the same weight. If an organisation must change in order to satisfy its goals, resources will be invested in making that change. We see an example of this in the requirement of a UK defence standard for safety-critical systems. This states that formal specifications should be produced and the system verified against these specifications. In order to maintain and develop their defence business, organisations are investing in formal methods training.

Without such organisational support, the introduction of new technology and methods is much more difficult to achieve. Formal methods are again a good example of this. There are sound arguments for the use of formal specifications [27] in software development. In spite of this, formal methods are rarely used except in the safety-critical systems domain. Although technical arguments against formal methods are put forward, we believe that the reasons why they have been rejected are because they fail to satisfy project management and organisational goals.

Project managers are concerned with completing projects successfully and are unwilling to take the risk of adopting radically new approaches without organisational support. If these approaches fail then the project management goals will not be achieved. Organisations may have business goals such as software quality improvement. However, they generally don't care about how this is achieved. It is quite clear that other approaches to quality improvement are effective and there is therefore no organisational support for formal methods.

The problem which is faced by new approaches to the software process is that they are likely to be perceived by engineers as oriented towards management goals as they guide and control the process. Management, however, may not see things in exactly the same way. Specialised notations and tools may be perceived by management as a new technologies which are a risk to their current development methods. Management

therefore see these as satisfying technological rather than managerial goals and are therefore unsympathetic to these changes.

To make an impact, therefore, proposers of software process change must have organisational support for that change. Organisations must be willing to invest time and money in promoting process change and provide an adequate training budget for engineers affected by the change. In one of our studies, it was notable that expensive CASE tools were unused. The reason for this was that tools were purchased from a capital budget while training was funded from a different budget. Although there was money to buy the tools, there was no money to pay for people to learn to use them.

3.3 Cultural Factors

Although they are very difficult to define in an objective way, cultural factors have an very significant effect indeed on whether or not changes which affect that culture are likely to be accepted. In the context of software processes, cultural factors influence the introduction of new processes and the modification and evolution of existing approaches to software development.

Cultural factors are significant at two different levels:

1. The organisational level where organisations develop their own distinctive culture which is recognised and (generally) accepted by the people working in these organisations.
2. The national level where different countries have different cultures. These cultural influences are less easily recognised by nationals of that country as they pervade all activities in that country. However, outsiders can often see immediately how national cultural factors influence the acceptance or otherwise of change.

At one extreme of organisational culture, there are very rigidly structured, hierarchical organisations where roles are clearly defined and individuals in these roles are not expected to overlap significantly with other roles. This culture, in the UK at least, is most apparent in large manufacturing organisations that have many years (sometimes more than 100) of engineering experience. The often-criticised 'waterfall' model of the software process is alive and well in these organisations and accepted without a great deal of criticism simply because its structured nature fits with the organisational culture.

At the other extreme, some organisations pride themselves and deliberately foster a culture of informality. Job titles may be deliberately vague and flexible allocation of work is encouraged. We see such a culture in many small companies and but also in larger companies which were founded as software companies in the 1960s and 1970s. More informal approaches, such as prototyping, are likely to be promoted in such cultures and rigid hierarchical models, rejected by them.

An organisational culture manifests itself most strongly in what are sometimes called 'high-reliability' organisations. These are organisations involved in safety-critical activities such as air traffic control. The importance of safety is pervasive in these organisations. Individuals in these organisations naturally give priority to safety irrespective of their role. This culture also pervades the software development process in these organisations so that there is a natural tendency to develop dependable processes.

At the national level, there are extreme differences between the cultures in Asia and the USA and less significant but nevertheless important differences between countries in Europe. The North America individualism is encouraged, job mobility (and hence technology transfer) is the norm and change is, in many organisations, seen as

welcome and necessary. By contrast, Isoda and Saeki [28] characterise Japanese social traits as:

1. Groupism. Individuals prefer to not to stand out from a group.
2. Gradualism. Radical change is unwelcome but gradual change is readily accepted.
3. Companyism. People are expected to work with the same company all their life.
4. Social rigidity. Roles and positions in a social hierarchy are well-defined and accepted.

At a European level, the differences between countries are less marked. Living within one such culture, it is difficult for us to be objective about this but, from a UK perspective, Germans have a reputation for thoroughness, French for abstraction, Italians for design flair, British for improvisation and so on. Of course, these are examples of cultural stereotyping but they do reflect real differences between the countries which pervade the educational and training systems. Those involved in software development will naturally adopt the local cultural traits.

There are very complex interactions between national and organisational cultures. A process which works in one part of a company may not be successful in another part of the same company which is based in a different country. These cultural interactions are increasingly important as software companies become multi-national. Process engineers in an organisation should be aware of them and sensitive to cultural issues.

In a consideration of computer science education Friedman and Kahn [29] stress that the linkages between the social on the technical are complex and are manifest in a variety of ways. In particular, they stress that cultural features are often manifest in terms of biases or presumptions which are designed into a constructed system. In particular they outline three forms of bias

- *pre-existing social biases*: where existing stereotypes and biases are embodied in the system. For example, the relationship between manager and subordinate may determine all actions have to be authorised.
- *technical biases*: where the solution of a technical problem in the construction of a system outlines a particular system behaviour. Consider for example the use of search strategies to find the most acceptable recipient of information.
- *emergent social biases* are manifest in use when social uses change and often require some reconsideration of the role of the systems in use. For example, changes in an organisation due to take over or merger can often result in an organisational culture which suggests that certain skills are significant yet the supporting computer system appears to devalue these skills.

These different effects work together to determine the effects of a system within a social setting. In the case of process modelling and evolution the chances of bias of some kind and the consequential effects are significant given that it seeks to represent the work people do.

3.4 Implications for software process research

Because of the complexity of organisational and cultural issues, there is no practical way in which they can be reflected in process support technology. What researchers must aim for, therefore, is the development of technology which is as culturally neutral

as possible. Given that these researchers live within a culture and will have their own cultural biases, this is a very difficult thing to do.

As process support technology reflects the way in which groups and organisations work, it is distinct from other CASE tools which embed the process models of how individuals should use particular design methods. This suggests that developing process technology products for the international market will be a difficult task. Developers of such products must examine them carefully for the inclusion of their own cultural biases. If these are significant, the products may be unacceptable in other countries and organisations.

Discovering cultural biases in software implies that, firstly, we need to develop education and training to make engineers aware of these biases and, secondly, that we need simple ways of analysing process models to detect possible cultural influences. Providing good process presentation tools and exposing the process description to as many people as possible is perhaps the most effective way of detecting such problems. When engineers respond with statements such as 'That wouldn't work here', there may be cultural reasons for this.

4. Conclusions

Rather than a report of specific software process research, this paper has discussed a number of social, organisational and cultural issues which must be considered when introducing software process technology. We believe that addressing these 'soft' issues is critical if process technology is to be successfully used.

We must admit that we have progressed much further in the area of problem identification than we have in suggesting solutions to these problems. However, from our research, we believe that developers of process technology should bear the following points in mind:

1. It is probably impossible to convince all members of an organisation who are involved in software development that they should follow a single process model. Professionals will always want (rightly, in our view) to demonstrate their professionalism and this means that they will have their own views on the software process.
2. Process descriptions should include facilities for attaching informal information which may provide critical process information. Support tools should recognise and present this informal information. It is not possible to decide in advance what entity types and relations may be required to represent social, human and organisational issues if, indeed, they can even be represented using such concepts.
3. Process support tools should pay more attention to presenting the process to end-users and developing method of involving these end-users in the preparation of process descriptions. It must be possible to leave part of the process 'undefined' and left to the judgement of engineers involved in that process. This implies that we have to take a 'lightweight' approach to process description with process details left to the development team.
4. We need more research into the human and organisational problems which arise when process technology is introduced into an organisation. The quantitative, process measurement approach suggested by process improvement models needs to be supplemented by qualitative studies which focus on the people involved in the process.

5. Acknowledgements

This work was partially funded by the UK Joint Council Initiative in Cognitive Science and HCI and by the European Commission in the REAIMS project (Project 8649).

6. References

[1] Osterweil, L., "Software Processes are Software Too". *Proc. 9th Int. Conf. on Software Engineering*, 1987. **2-12**.

[2] Hammer, M., "Reengineering Work: Don't Automate, Obliterate". *Harvard Business Review*, 1990. **July-August 1990**: p. 104-112.

[3] Suchman, L., *Plans and Situated Actions*. 1987, Cambridge: Cambridge University Press.

[4] Lehman, M.M. "Process Models, Process Programs, Programming Support". in *Proc. 9th Int. Conf. on Software Engineering*. 1987. Monterey, Ca.

[5] Curtis, B., H. Krasner, and N. and Iscoe, "A Field Study of the Software Design Process for Large Systems". *Comm. ACM*, 1988. **31** (11): p. 1268-87.

[6] Bentley, R., *et al.* "Ethnographically-informed Systems Design for Air Traffic Control". in *CSCW'92*. 1992. Toronto, Canada.

[7] Heath, C. and P. Luff. "Collaborative Activity and Technological Design: Task coordination in th London Underground control room". in *ECSCW'91*. 1991. Amsterdam.

[8] Sommerville, I. "Process Viewpoints". in *Proc. 4th European Workshop on Software Process technology*. 1995. Leiden, NL.

[9] Kellner, M., *et al.* "ISWP-6 Software Process Example". in *Proc. 6th Int. Software Process Workshop*. 1991. Hakodate, Japan.

[10] Longchamp, J., "An Assessment Exercise", in *Software Process Modelling and Technology,* A. Finkelstein, J. Kramer, andB. Nuseibeh, Editors. 1994, Research Studies Press: Taunton.

[11] Humphrey, W.S., "Characterizing the Software Process". *IEEE Software*, 1988. **5**(2): p. 73-79.

[12] Humphrey, W., T. Snyder, and R. Willis, "Software Process Improvement at Hughes Aircraft". *IEEE Software*, 1991. **8**(4): p. 11-23.

[13] Humphrey, W.S., "Software and the factory paradigm". *IEE/BCS Software Eng. J.*, 1991. **6**(5): p. 370-6.

[14] Davenport, T., *Process Innovation*. 1993, London: Ernst and Young.

[15] Feigenbaum, A.V., *Total Quality Control, 3rd edition*. 1991, New York: McGraw-Hill.

[16] Clements, A., "Computing at Work: Empowering Action by 'Low-Level' Users". *Comm. ACM*, 1994. **27**(1): p. 52-63.

[17] Greenbaum, J. and M. Kyng, *Design at Work: Cooperative Design of Computer Systems*. 1991, Hillsdale, NJ: Lawrence Erlbaum Associates.

[18] Le Quesne, P.N., "Individual and Oganisational Factors and the Design of IPSEs". *Comp. J.*, 1988. **31**(5): p. 391-7.

[19] Grudin, J., "Groupware and Social Dynamics: Eight Challenges for Developers". *Comm. ACM*, 1994. **37**(1): p. 92-105.

[20] Abbot, K.R. and S.K. Sarin. "Experiences with workflow management: Issues for the next generation". in *Proc. CSCW'94*. 1994. North Carolina.

[21] Warboys, B., "The IPSE 2.5 project: A Process Model Based Architecture", in *Software Engineering Environments: Research and Practice,* K. Bennett, Editor. 1989, Ellis Horwood: Chichester.

[22] Kadia, R. "Issues Encountered in Building a Flexible Software Development Environment". in *Proc. 5th ACM Symposium on Software Development Environments*. 1992. Tyson's Corner, Virginia.

[23] Ben-Shaul, I.Z., G.E. Kaiser, and G.T. Heineman, "An Architecture for Multi-user Software Development Environments". *Computing Systems*, 1993. **6**(Spring, 1993): p. 65-103.

[24] Canals, G., *et al.*, "ALF: a Framework for Building Process-Centred Software Engineering Environments", in *Software Process Modelling and Technology,* A. Finkelstein, J. Kramer, andB. Nuseibeh, Editors. 1994, Research Studies Press: Taunton, England.

[25] Montangero, C. and V. Ambriola, "OIKOS: Constructing Process-Centred SDEs", in *Software Process Modelling and Technology,* A. Finkelstein, J. Kramer, andB. Nuseibeh, Editors. 1994, Research Studies Press: Taunton, England.

[26] Hirscheim, R. and M. Newman, "Information Systems and User Resistance: Theory and Practice". *Comp. J.*, 1988. **31**(5): p. 398-408.

[27] Hall, A., "Seven Myths of Formal Methods". *IEEE Software*, 1990. **7** (5): p. 11-20.

[28] Isoda, S. and M. Saeki, "Software Engineering in Asia". *IEEE Software*, 1994. **1**(5): p. 63-9.

[29] Friedman, B. and P.H. Kahn, "Educating Computer Scientists: Linking the social and the technical". *Comm. ACM*, 1994. **27**(1): p. 65-70.