

The social construction of the software operation

Reinforcing effects in metrics programs

Helle Damborg Frederiksen, Jeremy Rose

Aalborg University, Department of Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark

Abstract

In a large software company in Denmark, much effort was expended capturing metrics about the company's software operation. The purpose of the metrics program was to change and improve the software operation. Writing software can be understood as a socially constructed practice, which can be analyzed using structuration theory. This structural analysis showed that the company's software operation followed an easily recognizable and widely understood pattern. The software operation was organized in terms of development projects leading to applications that then needed maintenance, and displayed a heavy focus on project development work and hitting the project deadline. Study of the metrics program (and the computer software underpinning it) revealed that the familiar pattern was also inscribed into the metrics software, heavily influencing the company's metrics practice. Rather than challenge the underlying social practice of the software operation, the metrics program reinforced it by adopting the same underlying values. Our conclusion is that, under these circumstances, metrics programs are unlikely to result in radical changes to the software operation, and are best suited to small, incremental improvements.

Keywords

Software process improvement, software metrics, social constructionism

Introduction

Many organizations make a conceptual separation in their software operation between the development of new software (usually cast as development projects), and maintenance of the applications that result from these projects. Maintenance activities often constitute the larger, and very much the less glamorous part of the total operation. 'For most large, long-lifetime software systems, maintenance costs usually exceed development costs by factors ranging from 2 to 4' (Sommerville 1992). Lientz and Swanson (1980) found that large organizations devoted at least 50% of their programming effort to maintaining existing systems, whereas Avison (1995) estimates that the effort given to maintenance can be as high as 60-80% of the total workload. However maintenance 'is often considered a burden and staff find the work unrewarding and often frustrating' (Avgerou and Cornford 1998). Sommerville also describes the 'negative image' of maintenance.

'Maintenance has a poor image amongst software engineers. It is seen as a less skilled process than program development and in many organizations is allocated to inexperienced staff.' (Sommerville 1992).

Project deadline pressures do not help this situation:

'the temptation is towards 'quick and dirty' solutions. The deadline for cutover may seem sacrosanct [sacred and inalterable]. It is politically expedient to patch over poor design rather than spend time on good design.....' (Avison and Fitzgerald 1995).

This conceptualization of the software operation (development projects leading to applications that then need maintenance) is also common in software houses – organizations dedicated to building software. The term 'software operation' refers to the analytical and programming work that leads to the development, upgrading and maintenance of software applications. These are the direct revenue generating operations, or 'operating core' (Mintzberg 1983) of a software house. Initiatives aimed at improving the software operation have in recent years been studied as

software process improvement (SPI) (Humphrey 1989; McFeely 1996) and often linked to the Capability Maturity Model (CMM) (Caputo 1998; Paulk et al. 1995). SPI focuses on continuous improvement of software development work practices. Assessment of current software operation capability normally leads to change strategies such as the improvement of project management practices like scheduling and tracking. CMM work focuses on software process standardization, and, as the level of maturity increases, process learning and improvement. However, experience shows that many SPI efforts fail to generate significant changes in organizational practice.

'Starting SPI is not difficult... However, turning assessment insights into action is the point at which many organizations fail. Others manage to initiate focused improvement projects, only to find that implementing new ideas is very difficult. Even when you succeed in implementing an idea in an individual project, you are still a long way from institutionalizing improvements.' (Mathiassen et al. 2001).

One strategy for learning about the software operation advocated by SPI theorists is a metrics program. "Software process data is gathered to learn how to make process improvement" (Humphrey 1989). A metrics program is the work of collecting and interpreting numerical data about the software operation (which is normally structured around a metrics software package), and feeding back the resulting understandings. A metrics program builds on measures, which each provide a quantitative indication of the extent, amount, dimensions, capacity, or size of some attribute of a software product or process (Pressman 2000). The measures are the result of the collection of one or more data points and they are related (often via algorithms built into the software package) to obtain indicators of performance. The measures and indicators are used to support management and improvement, either in specific projects or organization wide.

'Software metrics are initiated with the belief that they will improve software engineering and management practices. The rationale arises from the notion that you cannot improve something

without first measuring it.' (Gopal et al. 2002).

Establishing a well-functioning metrics programs is demanding. Difficulties may include:

1. Alignment of the measurement program with wider business and organizational goals,
2. Organizational commitment and resource sufficiency, and
3. The technical characteristics of the measurement program itself (Goldenson et al. 1999).

A study by Dekkers (1999) showed that nearly 80% of attempts failed within the first two years.

In view of the reported difficulties with SPI and metrics programs (particularly with respect to effecting lasting organizational change), it becomes important to ask 'what is the nature and extent of the improvement to the software operation that can reasonably be expected to result from a metrics program?' In this paper, we address this question by studying the metrics program at Software House (SWH - the name is anonymized). SWH is a large Danish software house, and their metrics program has been running for five years. A third party company, Waypoint (name also anonymized) provides a software package (Development Waypoint) for recording the data, and data analysis services which include benchmarking against leading software firms. The program has evolved to the point where, after considerable effort, reasonably reliable data is available about the software operation, and the company is currently seeking to improve the utilization of the data.

The frame of reference that we adopt when addressing the research question is that the software operation and the metrics program can be studied as socially constructed practice: that is evolving practice which is located in the shared understandings and actions of the people who undertake it, related to its context and historical development. This approach encourages a much wider appreciation of socio political- context than the focus on software process. The two socially constructed practices are related in as much as the purpose of the metrics program is to improve the software operation. Many theoretical lenses could be

adopted to study socially constructed practice; for example communities of practice (Wenger 1998), situated action (Suchman 1987), actor network theory (Callon and Law 1989; Latour 1987). Structuration theory (Giddens 1984) is preferred here because of its focus on the emergent effects of action and structure: an evolving pattern of behaviors and understandings. In this way we set out to study the emergent effects of one practice (the metrics program) upon another (the software operation); in particular the ability of the metrics practice to change ('improve') the software operation – its primary stated purpose.

Research method

This research project forms part of a collaborative practice study (Mathiassen 2000) which is intended to improve the use of metrics at SWH (see also Frederiksen & Mathiassen, 2002). The study, which began in October 2000, is sponsored by the company's director for software development, and operates under a company steering committee. The research team includes both researchers and practitioners, and one of the authors (Frederiksen) is a long term employee currently responsible for the metrics program.

The interpretive case study reported here was designed to help understand current practice. The information systems literature contains many examples of the interpretive case study (Brooks 1997; Jones and Nandhakumar 1993; Karsten 1995; Knights and Murray 1994; Sauer 1993; Walsham 1993; Zuboff 1988). The research style normally involves substantial involvement in the research situation in a participant or non participant, overt or covert mode, over a period of time. The goal of this style of research is 'deep familiarity' (Goffman 1989) with the research situation, resulting in 'thick' description which provides enough detail to allow analysis of the interpretations and inner thought worlds of the research subjects. The researcher cannot be assumed to be free of their own interpretations, and considerable care is needed to offer analysis in an open and explicit manner, so that the reader is able to make judgments about its validity. Walsham (1995), following Eisenhardt (1989), suggests that theory may be involved in interpretive case

studies in three different ways:

1. As an initial guide to design and data collection
2. As part of an iterative process of data analysis
3. As a final product of the research.

The iterative approach used here allows the theoretical position to be developed as data collection and analysis proceed. There may be a problem in generalizing from case study research, where depth is substituted for breadth, but Walsham suggests that four types of generalization are possible.

1. Concepts may be developed (such as Zuboff's 'informate')
2. Theory may be generated (as with the structurational model of technology (Orlikowski 1992))
3. 'Specific implications in particular domains of action' may be drawn (such generalizations are often formulated as tendencies rather than predictions)
4. Less focused learning may be termed 'rich insight.'

Data collection for the study included ten semi-structured interviews with management, metrics staff, quality assurance staff, and software engineers. The participants were chosen to represent different interests in the collection, interpretation, distribution, and usage of metrics data, e.g. software senior manager, software project managers, team manager, metrics controller, general controller, manager with responsibility for quality assurance and project planning and tracking in SAP, manager with responsibility for using data for software process improvement, software engineers, and maintenance engineer. Interviews lasted for three hours, were documented and corrected and approved by the interviewee. We studied plans, decision reports, and e-mail correspondence related to metrics, minutes of meetings from the metrics staff, consultant's reports and documents from the organization's process improvement initiative and the company intranet. This constitutes the formal data collection process. However, Frederiksen's role as practitioner in the research situation allows a daily familiarity with the company and the metrics program which goes beyond participant observation or the temporary engagement

generated by action research (for instance, a complete five year record of all emails sent to her in connection with the project was available for study). This deep engagement with the research situation was recorded in a personal log for the duration of the study period.

An important part of the research process involved achieving a balance between distance and engagement (Nandhakumar and Jones 1997). Engagement promotes understanding, but can lead to bias and over-subjective interpretation. Distance helps with achieving defensible understandings, but can lead to superficiality. In this project, one of the authors was heavily engaged with the research situation, and the other with the theory. Balance was achieved by a series of long analytical conversations, in which each researcher took care to transfer their engaged understandings to the other, and acted as uninvolved (distant) critic for the understandings they received. All analytical insights into the research situation were assumed to be provisional and in need of validation. Initial data collection was followed by structurational analysis, leading to the generation of insights, followed by a further round of data collection targeted at those insights and further analysis. The insights generated from this round of analysis were taken back to the interviewees and project sponsors for validation. The data and case study material (section 4) is separated from the analysis (section 5) as far as possible in the writing so that the reader can judge the validity of the analysis and conclusions. The research is overt and conducted with the encouragement of the company concerned. The end product of the of the research process can be characterized as specific implications in a particular domain (the use of software metrics programs for software process improvement).

Structuration theory and the social construction of practice

Anthony Giddens' structuration theory (expressed in 'The Constitution of Society' (1984)) has been used in the study of information systems for some time (Barley 1986; Brooks 1997; Jones and Nandhakumar

1993; Karsten 1995; Orlikowski 2000; Orlikowski 1992; Orlikowski and Robey 1991; Rose and Scheepers 2001; Walsham 1993). There are already three published reviews of this literature (Jones 1997; Rose 1998; Walsham and Han 1991) and all the reviewers note the use of structuration theory for analyzing empirical situations involving information systems.

Structuration theory offers an explanation of social practice as the recursive interaction of agency and structure. Agency refers to what human actors choose to do, whereas structure refers to the sets of personal and collective understandings which form the context in which they do it, and partly determines what they are able to understand as a viable potential action. Human agency, in Giddens formulation, is the 'capacity to make a difference' (Giddens 1984 pp 14) - (also known as 'transformative capacity'). He defines structure as

'rules and resources recursively implicated in social reproduction; institutionalised features of social systems have structural properties in the sense that relationships are stabilized across time and space'.....{Structure} 'exist only as memory traces, the organic basis of human knowledgeability. Structure refers, in social analysis to 'the structuring properties allowing the 'binding' of time space in social systems, the properties which make it possible for similar social practices to exist across varying spans of time and space.' (Giddens 1984 pp 17).

Social practice can be regarded as the interaction of structure (sets of individual and communal understandings) and action. Giddens recasts the two independent sets of phenomena (dualism) of structure and agency as a 'duality' - two concepts which are dependent upon each other and recursively related.

'The structural properties of social systems are both medium and outcome of the practices they recursively organize' (Giddens 1984 pp 25).

Structuration is thus the process whereby the 'duality of structure' evolves and is reproduced over time space to constitute social practice. Giddens suggests that the duality of structure can be analyzed as dimensions including signification, domination and legitimation (structure) and the related concepts

communication, power, and sanction (interaction).

The social construction of practice (in this case software operation and metrics program) can therefore be understood, in structural terms, as the recursive interaction of structure and agency (analyzable in terms of the dimensions above), reproduced over time and space and integrated into emergent patterns.

SWH: the software operation and the metrics program

Background

The business of SWH is to develop and maintain IT-solutions, including applications for payroll, financial administration, and budgeting, for large public clients in Denmark. Lately, SWH has focused on delivering web applications for private citizens to use, such as applications for accessing and updating information on the web. In April 2001 SWH had 2355 employees, 620 of whom were software developers. This makes SWH one of the largest software organizations in Denmark. SWH is distributed widely across Denmark at 12 sites, and the software organization is represented at four sites. Though the company's customers may choose their software solutions freely from the market, in some domains (such as tax administration) SWH has, at least for the time being, the leading market position. Customers very rarely pay directly for development work. According to the chief accountant at SWH the revenue comes from licensing arrangements, which guarantee maintenance and support for the applications. In 2001 less than 1% of the development projects were charged to customers on an hourly basis.

The software operation at SWH

SWH is organized in a conventional hierarchical divisional structure, and the software operation is organized in teams according to the applications, (for example a team is responsible for the tax application). The application teams are responsible for correcting faults, supporting the users and for upgrading the applications with additional functional requirements and more up-to-date technology platforms. New

software development is organized in projects, with project members drawn from different departments as appropriate. Because of the wide potential client base of the software applications, users are often represented on development projects by company experts with local knowledge, rather than being directly involved. Maintenance programmers are dedicated to correcting errors, supporting users, and programming functionality that is not part of the projects. In 2001 one fourth of the software operation was dedicated to developing new applications. The remaining part was dedicated to the maintenance of the existing applications.

Perceptions of project and maintenance work are very different. "There is more prestige associated with working on projects than with working on maintenance" says a software engineer with nearly twenty years experience working across the organization (interview, software engineer, September 2001). Project work is perceived as "the most exciting and challenging" with "stimulating new worlds and "technical challenges to be overcome... new recruits prefer to work here" (interview, maintenance engineer, June 2002). Maintenance work is "little valued and boring." Maintenance programmers have "security without changes ... a stable environment - you know what to expect. There's no particular reason to get better educated or develop. In the short term, the work can be successfully done without it. However there's a risk that you'll pay for this later... furthermore you're locked into the job. Productivity can be very low. It's measured by the fault statistics. Nobody thinks whether it would be better to add new functionality. There's a one-sided focus on reducing the fault count" (interview, maintenance engineer, June 2002). Poor perceptions of maintenance sometimes mask its real importance. Maintenance workers are "key personnel - knowledge is power...remember that this is where we earn money, so it must really be important...you're thought of as flexible, if you agree to stay in maintenance. Some people get a good salary to stay... the average age is higher here" (interview, maintenance engineer, June 2002). There is a structural divide between development work, organized in projects, and maintenance work,

which comes later. "It's the common perception at SWH that everything that comes after a project's specified deadline is maintenance" (interview, software engineer, September 2001).

There is no compulsory method for software development (project leaders and teams are free to choose their own project organization and development methods), but, by unspoken convention, projects always adopt some kind of waterfall model structure. Therefore the language of analysis, design, testing, implementation, rollout and maintenance is standard, and can be seen, for instance throughout the various development guides issued by the methods department. The status of projects is monitored using the familiar traffic light convention. A project's status can be green amber, or red. However the determination of a project's status is not always simple. An experienced project manager was specially selected for a difficult project (project A) (interview, project manager, March 2001). Project A ran into trouble early on, and the project manager recognized this. She also knew her reputation partly depended on hitting the deadline. She told her managers about the difficulties, but they didn't want to listen. She thought the project should be red-lighted as a signal that it was in trouble, but the managers would not agree, knowing that this was a sign of failure that they could be held responsible for. She proposed a reduced functionality plan that would see the project through to deadline, but it was rejected. Project A went through many crises, demanded much overtime, and its functionality eventually had to be cut along the lines that the project manager had suggested.

In the management of projects, the most important consideration is hitting the deadline. According to one senior manager "the expert board {charged with overseeing the development projects} concentrates only on projects keeping to schedule - they have no responsibility for business aspects." According to the manager responsible for the expert board, "its objective is to make sure that projects run to schedule, but we also keep an eye out for functionality and quality, and we also act as a sparring partner for project managers." However another senior manager (much closer to development work) commented "the expert

board is interested in just one thing – hitting the deadline.” This has resulted in a widespread understanding of the deadline imperative. At a series of courses for trainee project managers (developers, technicians, sales people, implementers drawn from all parts of the organization), the trainees were asked to choose between deadline, cost and quality as the most important success factors. All twenty trainees chose deadline.

The management of projects displays a number of ways of manipulating the projects in order to hit, or apparently to hit deadlines. In one recent project, (project B) organized along a traditional analysis-design-implementation-testing-rollout pattern, it became obvious that the deadline could not be met (interview, quality assurance personnel, October 2000). The solution was to organize another, separate project for part of the testing and the rollout. The outputs of the first project became the starting point for the next project. In this way the formal requirements for meeting the deadline for the first project could be achieved, at least in appearance. Skipping part of the testing, or documentation in order to meet the deadline is also fairly commonplace. Another method for hitting the deadline is to reduce the program functionality, as happened in project A (reported above). The missing functionality can be then be rescheduled as another project, or (quite commonly) handed over to the maintenance engineers and undertaken as part of maintenance duties.

This focus on deadlines can sometimes be at the expense of software quality. “That’s also one of the problems I’ve experienced – that the managerial focus on quality has not been huge” (interview, experienced software middle manager, January 2002). Quality, in the language of SWH, refers to the performance of the code that has been written – its ability to execute cleanly without bugs and its functional ability to fulfill users’ needs. Quality is usually measured by fault statistics (the complaints registered by users). Quality considerations will not necessarily affect a project’s status, if it appears to be running to schedule. “A project can easily have big quality problems (for example in a system test) and still be green” (interview, experienced software middle manager, January 2002). This can result in a

problem at release time, as imperfectly tested code is rushed into operation to meet the deadline. “It’s common to experience a boom in faults at release time” (interview, middle manager 1, November 2000). Apparently successful projects can lead to problems later - “we have many examples of ‘exemplary’ development projects which brought praise to everyone in the project group, where there was a big problem later in maintenance with the number of bugs, the system’s operational situation” (interview, experienced software middle manager, January 2002). The structural divide between development and maintenance means that there is little feedback – “nobody follows up where the bugs arose – which project group produced the bugs.....nobody ever talks about it. In any case, the project group has moved on to the next job” (interview, maintenance engineer, June 2002).

The metrics program at SWH

SWH has a long tradition for collecting simple measures, for example the number of defects in an application. In January 1997 the metrics program was implemented with the stated intention of increasing productivity and quality, through measuring key indicators and benchmarking against other companies. ‘One measurement is better than a thousand opinions,’ (interview, senior manager, October 2000). The insights available from the metrics program should lead to change in the company’s working practice. “Metrics and benchmarking are pointless by themselves. It’s the resulting change that’s the important thing. Metrics are the means with which to actively achieve change” (minutes, metrics program pre-project group, Nov. 1996). “The purpose of the metrics is to realize efficiency gains through making costs visible, and identifying improvement areas” (presentation, CEO, Jan 1997).

An external company (Waypoint) supplied the metrics program. Data is collected according to the model supplied by Waypoint (Figure 1) and entered into Development Waypoint, the software supplied for this purpose. The software is built around a database with fields organized according to the model. At the beginning of the project the project manager reports on (amongst other things) the actual start date, name of project manager, data on estimates

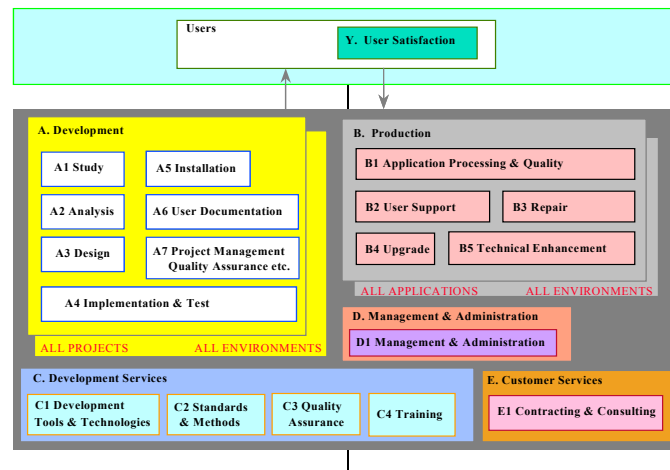


Figure 1: Development Waypoint

and so on. They estimate the number of function points (a measure of program complexity) in the project. “Counting function points is today reckoned the only recognized way to compare project complexity” (meeting minute, project group, Jan 1997). Through the life of the project other data such as person hours and cost are reported every quarter, and further data are reported upon completion. In the model (Figure 1) the measures from C (development services), D (management and administration) and E (customer services) are regarded as overheads to the software operation, which Waypoint represent as A (development) and B (production). SWH do not use the term ‘production’ and in their internal documents it is relabeled ‘maintenance’. According to the model, development is divided into the traditional activities: study, analysis, design etc. Production is viewed in terms of the activities: user support, repair, upgrade (new functionality), and technical enhancement. SWH chose not to use the customer satisfaction (Y) part of the model. The data are processed, validated and packaged at SWH, then sent to Waypoint for analysis. The analysis results include indicators, benchmarked against successful companies in the industry, both at a general level and at a project and application level. Once a year Waypoint delivers a written report including an analysis of strengths and weaknesses pf the software operation and a set of recommendations for further actions.

If we relate the Waypoint indicators to the software operation at SWH, then A1-A7 measures relate to projects, B1 measures relate to the quality of the resulting application and B3 measures relate to maintenance. B4 (upgrade) and B5 (technical enhancement) might refer either to projects or maintenance in the SWH software operation. Of the measures implemented by SWH, 64 refer to projects and 37 to applications and maintenance. In the model’s graphical representation, development is the largest, so presumably most important activity area. Waypoint analyzes the data annually and provide two levels of reporting: a management summary and a much more detailed numerical report, in which quantitative indicators are cross-tabulated against categories derived from descriptive indicators. The management summary focuses on eight aggregated reporting indicators. Project indicators dominate; furthermore one of the four project indicators relates directly to the project deadline, and two more are indirectly related since they are dependent on time measures.

We should also observe some things which are missing from the metrics program. Income plays no part, so it is not possible to relate income from the software operation to the costs of particular applications and the projects that developed them. Neither is it possible to relate a development project to a specific application and its subsequent maintenance. This means that it is impossible to link specific project

measures with measures concerning the applications that result from the projects, or the later maintenance work that might result partly from poor (or good) design and programming. Thus if a project was completed on time, but at the expense of application quality, and this later caused many maintenance problems, there would be no way of learning this from the metrics program. Finally, SWH chose not to collect user satisfaction metrics (Y in Figure 1). Hence, there are no measures of application quality as assessed by the firm's customers in the metrics program.

However, "interpreting results coming from the metrics program can be quite difficult... you need a deep understanding of the Waypoint model and function points to interpret the data" (minute, project meeting, March 2002). It has also proved difficult to target relevant conclusions from the report at appropriate company members: "in order to interpret the Waypoint results at a project or site level, its necessary to have insight both into to the function point method and the Waypoint model, and also to understand other completed SWH projects. You cannot expect this from the target groups. And the higher up the organization the report goes, the less insight you can expect" (personal note, Waypoint controller, June 2002). Some were openly suspicious of the results - "the indicators (graphs and tables) give more noise (mistrust of the function point method, mistrust of consistency in data collection between sites, fueling of rumors) than food for careful consideration" (personal note, Waypoint controller, June 2002). Others questioned the relevancy of the results- "it hasn't been possible to find graphs in the Waypoint reports which are relevant for project managers and employers" (personal note, middle manager 2, March 2002). "If this is really useful, you should tell us how, its got no value for the individual project manager" (personal note, middle manager 2, March 2002). Some denied that the program was useful or necessary - "the metrics confirm what we already knew.....we don't need metrics to improve our processes.....SWH's culture encourages improvement anyway" (interview, middle manager 1, April 2002)

Some of the reporting and utilization of metrics

is quite political. The allocation of function points, for example, causes some argument at SWH. "Employers would like to see for themselves that the function points numbers are OK, and that the function point numbers give a realistic picture of a system's complexity" (e-mail, senior manager 1, March 2000). "Function points, as a measure of size, are regarded as unjust" (personal note, senior manager 4, May 2002). If engineers didn't agree with the allocation of function points, they were unlikely to value the results that later came from the metrics program. "Function points contradict peoples' intuition.....employees should participate in the setting of function points, in order to commit to them" (personal note, middle manager 2, March 2002). Function point allocation is not a precise science, but is nevertheless necessary in order to compare development projects of different size and complexity. As another example of political dispute, it is not easy to say on which day a project started. Should this be the day the project was approved, the day the project manager was appointed, the day the first engineer started work, or the first day on site? A project manager would be wise to record the start date as late as possible, and the number of function points as high as possible, since this will improve all the major indicators for their project. Software managers were not particularly interested in the metrics program to begin with, regarding it as a distraction, but later learned that their projects (and therefore their work) could be evaluated through the program. Projects supervised by one particular manager (who was more than usually involved with the metrics program) began to show significantly better in all the indicators. Other managers, however, wanted his count of function points checked. They questioned whether his projects actually were better, or whether he had simply learned how to report the measures in a favorable way. Where metrics were intended to highlight problems, managers could sometimes be seen defending their own territory - "the development process runs smoothly. Problems should be sought in other parts of the organization" (email, middle manager 1, April 2002).

In 2001 the data available from the metrics program showed that, benchmarked against

good industry performers, SWH performed comparably well on the aggregated timeliness indicator – the projects did indeed hit their deadlines. However most of the other indicators were lower than the benchmarks. Despite five years of the metrics program it was difficult to point to areas where the program had made a significant impact on the company's procedures "Recommendations made on the basis of metrics are correct and relevant, but they don't necessarily lead to an actual change in behavior" (personal note, middle manager 1, April 2002). This was also perceived as a danger - "if the data is not continually used and seen to be used, the metrics program will simply lead a life of its own" (minute steering committee, senior manager, April 2001).

The social construction of the software operation at SWH – a structural analysis

The software operation

In terms of the software operation, structure primarily means the social and organizational arrangements in place (formal and informal), which both enable and constrain the writing of code. These constitute a set of rules and resources which actors employ in their software writing process. Agency is then the process of employing those rules and resources in the writing of software. In a development project, for instance, programmers employ individual and collective understandings of how a development project should be carried out (some of which are formalized as organizational procedures and some of which are part of their every day understandings), and resources such as budgets and development tools, in the writing of software. However the development project is an instance of software writing in which understandings are put in to action and realized, with varying degrees of success and fulfillment of expectations. It therefore contributes to individual and collective understandings, confirming or changing them in the process. Social construction of the software operation therefore refers to the recursive process whereby software is written under a set of social and organizational arrangements. The signification

structure (the shared meaning or mode of discourse) of the software operation at SWH is understood in terms of development projects (which result in applications or parts of applications), and maintenance of the applications. This is reflected in the structure of the company, the language ("maintenance" (SWH) not "production" (Waypoint)) methods department guides and so on. Substantial programming efforts (an upgrade to meet a change in legislation, for instance) may be cast as projects, even though they are not 'development' in the sense of developing an application from scratch. This is to say that the writing of code is understood either as development or maintenance, even though the actual activities concerned may be quite similar ("everything that comes after a project's specified deadline is maintenance"). Development is understood to be a project, and a project is understood to follow a waterfall model, that is to be composed of a linear set of activities including analysis, design, testing and implementation. The legitimation structures of the company ensure that projects are seen as more important than maintenance (see for example the maintenance engineer's description of her work) although maintenance is considerably more of the workload, and important in the financial structures of the company ("this is where we earn the money"). In project work, hitting deadlines is the most important goal and measure of success ("interested in just one thing – hitting the deadline"). Structures of domination in the company mean that programmers who are perceived to be more skilled become part of the elite who work on development projects and later become project managers. Successful project managers become senior managers who have the authority to perpetuate the same values, and exclude those with different perspectives (someone with a background in sales, for example). Other programmers work primarily on maintenance, sometimes many years on the same application, without wielding much formal power in the organization, though the software operation is highly dependent on their skills, experience and knowledge ("key personnel"). Users are often locked into legacy systems and consequently wield little consumer power.

Enduring structures favoring projects and hitting project deadlines over other features of the software operation provide the context for specific instances of practice. Managers can be seen exercising power over the 'red-lighting' of project A. Projects that fail or miss deadlines mean loss of face and reputation, and possible sanctions in terms of career progression, so the managers concerned come into conflict. The project manager tries to avoid being held responsible for missing the deadline, whilst her managers try to avoid being seen to be in charge of a failing project. In doing so they confirm the importance of the structures they work within, and reenact them. The deadline's the important thing. It is part of the programmers' understanding that the software they build should be thoroughly tested before being released to clients, and also part of their shared underlying model of development, but the project manager understands that deadline is more important, and has the power to overrule other members of the development team. In doing so s/he helps set expectations for the next project. Similarly, when project managers decided to cut the functionality of a system (as in project A) they prioritize the deadline over other legitimate expectations. According to the waterfall model interpretive scheme the coding of functions specified in a requirements specification is development work, but unwritten norms in the company allow it to be re-labeled as maintenance when the deadline is at stake. In this way the social practice of the software operation is constructed and re-constructed over time.

Thus one part of the software operation (new developments in the form of projects) is viewed as very important, whilst the much larger maintenance operation is viewed as less important. With project work one measure of performance, the project deadline, becomes more important than other measures (such as application quality – "managerial focus on quality has not been huge"). Actions that project staff take reproduce this legitimation scheme. Developers demonstrate that they are technically skilled by working on prestigious projects ("stimulating new worlds.....technical challenges to be overcome"), and position themselves so as to be seen to be hitting project deadlines, at least nominally, because they know

their reputations depend upon it. Successful actors (who understand the legitimation system and position themselves well) become the next generation of the powerful, who are in a position to reinforce the existing structures.

The metrics program

When we look at the metrics software depicted in the model provided at Figure 1, we find that it shares the same view of the software operation. The majority of the metrics and reporting indicators concern the software development projects; only a few indicators relate to the application quality and maintenance work. Within the project measures, timeliness (effectively deadline) measures play the most significant part. Reporting from waypoint shares the same focus. This is not very surprising, indicating that the developers of the software package shared the same perspective, and to some extent, inscribe their value system into the package as they build it. It is also necessary that the metrics package reflects the structure of the software operation in order to usefully collect data.

The implementation of the metrics program was seen as an opportunity to change the software operation practice ("it's the resulting change that's the important thing"). It makes visible certain parts of software practice, giving an opportunity to change shared structures of understanding and engineers' actions. Data is collected by engineers, but the program was championed by the Chief Executive Officer, and reports go to senior managers, underlining the domination structure. Certain aspects of the software operation are made visible, legitimizing them, and prioritizing them over other aspects which are not measured. Thus the collection of data about (for instance) project start and finish dates signals that these are considered important by the sponsoring senior management. Conversely a decision not to collect metrics (about user satisfaction, for instance) may send a signal that these aspects are less important. However, metrics program managers made few decisions about what data to collect. They chose, entirely reasonably, to largely follow the course set by the designers of the Waypoint software package, but by doing so unintentionally implied that they share the Waypoint value scheme. Managers and

engineers find ways to legitimize their positions by understanding the metrics process and ensuring that their own indicators are presented in the best possible light (for example in the discussions and interpretations of function point allocation). Differing semantic interpretations of the terms used (for instance in the social construction of what a 'function point' represents) make this possible even before the actors begin to 'cheat' (for instance by not providing data about badly performing projects which they know should be reported). It also means that 'losers' in the metrics situation (managers with poor metrics) may accuse 'winners' of 'cheating'. Senior managers are understood to be overseeing more junior colleagues and engineers (reinforcing structures of domination); checking whether they are performing well in selected areas which are thereby deemed important. Managers can be seen in rivalry over what signals about their personal projects are sent to senior management via the medium of the metrics program. Managers with favorable indicators are challenged, or the program invalidated ("we don't need metrics"). Maximizing one's position is a natural part of organizational life; however, in order to do this managers must focus on projects and project deadlines, because this is built both into the metrics model and to organizational life. In the five years of operation, the metrics program became a part of SWH's organizational routine, but there was little evidence that there were significant changes to the software operation as a result. The language of 'function points' became accepted and well-understood (and sometimes regarded as synonymous with the metrics program), but the concept then became part of well-entrenched norms and values concerning projects. No similar effort was evident to change the signification structure of the maintenance programmers' work. This metrics program focus on projects and project deadlines feeds back in the structurational cycle of the software operation and reinforces it.

In the evolution of the social practice of the software operation (that is its routinization and continuation over time and space), the metrics program has largely served to reproduce existing patterns, rather than to transform them into something recognizably different. The

focus on projects and deadlines was inscribed into the Waypoint software package, this pointed the way for the metrics program which was easily adapted to fit with the company's underlying software operation practice. Although the program was intended to change the software operation, many of the signals sent, and the actions taken in collecting metrics tended to reinforce the existing structures of signification, legitimation and domination. With the benefit of hindsight, it is doubtful that this reinforcement process could have been avoided.

Conclusions

This paper has argued that the software operation and metrics program at SWH can be understood as linked socially constructed practices, and analyzed with structuration theory. The analysis shows that the SWH software practice falls into a familiar pattern of development and maintenance, but has two particular characteristics: it prioritizes project development work over maintenance, and hitting the project deadline over other project success indicators (such as application quality or user satisfaction). Examination of the metrics software showed that it was also heavily influenced by these priorities, emphasizing the collection of data about projects and deadlines; whilst the reporting practice of the third party metrics company showed the same emphasis. In adopting the metrics program more or less as it was supplied, SWH developed social practices around the collection and use of metrics which tended to reinforce the existing priorities, rather than challenge them.

However, it is not necessarily the case that the focus on projects and project deadlines is beneficial to the company. Much of the work of the company, measured by programmers' time or income structure is maintenance, and there is a relationship between the quality of the applications that are developed and the resulting maintenance that needs to be undertaken. As researchers we may speculate that refocusing the company's energy on writing code which is well-tested, bug free and meets customers' needs, and targeting maintenance for improvement efforts might produce better

overall improvement for the company than the present focus on projects and deadlines. Though this remains a hypothesis, we can conclude that this is exactly the kind of challenge to social practice that the metrics program will never raise. The program is unsuited to producing radical improvements to existing practice – in the words of one manager “it only tells us what we know already.” In other words, the company’s metrics analysis takes place within the Waypoint frame of reference, which is very similar to SWH software practice. Experience with five years of metrics analysis in the company has shown it does not result in big changes. The company has slowly but steadily approached the benchmarks (part of this improvement may be attributed to learning how to report favorably), leading to a feel-good factor which contributes a further reinforcement effect. One metrics specialist summarized the situation neatly: “our real problem is: if our problem is anything except project deadlines, we can’t discover what it is.” Meanwhile the program becomes further integrated into the existing social practice. Shortly before the time of writing the chief executive announced an initiative to ‘increase timeliness and productivity within projects’ Baselines and improvements will be measured through the metrics program.

In reflecting upon the nature and extent of change that can be expected from a metrics program, we conclude that metrics programs are not neutral, but reflect perspectives on software operation commonly held amongst developers. Companies naturally choose or develop programs which reflect their own social construction of the software operation, and the programs therefore produce hidden reinforcing effects which tend to support existing practices rather than challenge them. Metrics programs can easily be integrated into the software operation’s cycle of structure and action, reinforcing the dominant logics rather than changing them. This does not mean the programs are without value, but rather that they will tend to produce small incremental improvements to existing practices. Challenging insights and radical improvement proposals must therefore come from other directions. However an individual armed with such a radical change proposal might be able to

use carefully selected and targeted metrics as part of a strategy to gradually alter the social construction of the software operation.

Acknowledgements

This research was in part supported by the Danish National Centre for IT research. Thanks to all the colleagues at SWH who were generous with their time, and colleagues at IRIS who reviewed an earlier version of the paper.

References

- Avgerou, C., and Cornford, A. *Developing Information Systems*, MacMillan, Basingstoke, 1998.
- Avison, D. E., and Fitzgerald, G. *Information System Development: Methodologies, Techniques and Tools*, McGraw-Hill, Maidenhead, 1995.
- Barley, S. R. “Technology as an Occasion for Structuring: Evidence from Observation of CT Scanners,” *Administrative Science Quarterly* (31), 1986, pp. 78-108.
- Brooks, L. “Structuration theory and new technology: analysing organisationally situated computer-aided design,” *Information Systems Journal* (7), 1997, pp. 133-151.
- Callon, M., and Law, J. “On the Construction of Sociotechnical Networks,” *Knowledge and Society* (8), 1989, pp. 57-83.
- Caputo, K. *CMM Implementation Guide: Choreographing Software Process Improvement*, SEI series in Software Engineering, Addison-Wesley, Reading, MA, 1998.
- Dekkers, C. A. “The Secrets of Highly Successful Measurement Programs,” *Cutter IT Journal* (12:4), 1999, pp. 29-35.
- Eisenhardt, K. “Building theories from case study research,” *Academy of Management Review* (14:4), 1989, pp. 532-550.

- Frederiksen, H. D., and Mathiassen, L. "Diagnosing Metrics Practice in a Software Organization" in *New Perspectives on Information Systems Development: Theory, Methods and Practice*. Harindranath, G., Rosenberg, D., Sillince, J. A. A., Wojtkowski, W., Wojtkowski, G., Wrycza, S. and Zupancic, J (eds.), Kluwer Academic, New York, 2002.
- Giddens, A. *The Constitution of Society*, Polity Press, Cambridge, 1984.
- Goffman, E. "On Fieldwork," *Journal of Contemporary Ethnography* (24:2), 1989, pp. 123-132.
- Goldenson, D. R., Gopal, A., and Mukhopadhyay, T. "Determinants of Success in Software Measurement Programs," *The Sixth International Symposium on Software Metrics: "Taking the Measure of New Technology"*, Boca Raton, Florida, 1999.
- Gopal, A., et al. "Measurement Programs in Software Development: Determinants of Success," *IEEE Transactions on Software Engineering* (28:9), 2002, pp. 863-875.
- Humphrey, W. S. *Managing the Software Process*, Addison Wesley, Reading, Massachusetts, 1989.
- Jones, M. "Structuration and IS" in *Re-Thinking Management Information Systems*. Currie, W. L. and Galliers, R. D (eds.), Oxford University Press, Oxford, 1997.
- Jones, M., and Nandhakumar, J. "Structured development? A structural analysis of the development of an executive information system" in *Human, Organisational and Social Dimensions of Information System Development*. Avison, D. E., Kendall, J. E. and DeGross, J. I. (eds.), North-Holland, Amsterdam, 1993.
- Karsten, H. "It's like everyone working round the same desk.' organisational readings of Notes, *Scandinavian Journal of Information Systems*," 7 (1:), 1995, pp. 7-34.
- Knights, D., and Murray, F. *Managers Divided: Organisation Politics and Information Technology Management*, Wiley, Chichester, 1994.
- Latour, B. *Science in action: how to follow scientists and engineers through society*, Harvard University Press, Cambridge, MA, 1987.
- Lientz, B. P., and Swanson, E. B. *Software Maintenance Management*, Addison-Wesley, Reading MA, 1980.
- Mathiassen, L. "Collaborative Practice Research" in *Organizational and Social Perspectives on Information Technology*. Baskerville, R., Stage, J. and DeGross, J. (eds.), Kluwer Academic Publishers, Boston, 2000, pp. 127-148.
- Mathiassen, L., Pries-Heje, J., and Ngwenyama, O. *Improving Software Organisations; from Principles to Practice*, Addison-Wesley, Boston, 2001.
- McFeely, B. *IDEAL: A Users' Guide for Software Process Improvement*, SEI, Pittsburgh, 1996.
- Mintzberg, H. *Structure in Fives*, Prentice-Hall, Englewood Cliffs, 1983.
- Nandhakumar, J., and Jones, M. "Too close for comfort? Distance and engagement in interpretive information systems research," *Information Systems Journal* (7), 1997, pp. 109-131.
- Orlikowski, J. "Using Technology and Constituting Structures: A Practice Lens for Studying Technology in Organizations," *Organization Science* (11:4), July-August 2000, pp. 404-428.
- Orlikowski, W. J. "The Duality of Technology: Rethinking the Concept of Technology in Organizations," *Organization Science* (3:3), 1992, pp. 398-429.
- Orlikowski, W. J., and Robey, D. "IT and the Structuring of Organizations," *Information Systems Research* (2:2), 1991, pp. 143-169.
- Paulk, M. C., Weber, C.V. Curtis, B. and Chrissis, M.B. *The Capability Maturity Model: Guidelines*

for Improving the Software Process, SEI Series in Software Engineering., Addison-Wesley, Reading, MA, 1995.

Pressman, R. S. Software Engineering - A Practitioner's Approach, fifth edition McGraw-Hill, London, 2000.

Rose, J. "Evaluating the contribution of structuration theory to the information systems discipline," 6th European Conference on Information Systems, Aix-en-Provence, Euro-Arab Management School, Baets, W. R. J. (ed.), 1998, pp. 910-924.

Rose, J., and Scheepers, R. "Structuration theory and information systems development; frameworks for practice," European Conference on Information Systems, Bled, Slovenia, Smithson, S. and Avgerinou, S. (eds.), 2001.

Sauer, C. Why Information Systems Fail, a case study approach, Waller, Henley on Thames, 1993.
Sommerville, I. Software Engineering, Addison-Wesley, New York, 1992.

Suchman, L. A. Plans and Situated Actions: The Problem of Human-Machine Communication, Cambridge University Press, New York, 1987.
Walsham, G. Interpreting Information Systems, Wiley, Chichester, 1993.

Walsham, G. "Interpretive case studies in IS research: nature and method," European Journal of Information Systems (4), 1995, pp. 74-81.

Walsham, G., and Han, C. K. "Structuration Theory and Information Systems Research," Journal of Applied Systems Analysis (17), 1991, pp. 77-85.

Wenger, E. Communities of practice: learning, meaning, and identity, Cambridge University Press, Cambridge, U.K.; New York, N.Y., 1998.

Zuboff, S. In the Age of the Smart Machine, Basic Books, New York, 1988.