The Soft System Methodology as a Framework for Software Process Improvement

Peter D.C. Bennetts
Cheltenham and Gloucester College of Higher Education

A. Trevor Wood-Harper
University of Salford

Stella Mills
Cheltenham and Gloucester College of Higher Education

The overall aim of this paper is to argue that only a systems-based approach to information systems development (ISD) and software process improvement (SPI) are likely to cover all the recognised issues. This paper will examine reports in the literature concerning problems associated with ISD and implementation. The characteristics of these problems are identified. It is noted that these problems have been known for many years. The responses to the problems fall into two broad areas which are characterised as paradigms. On the one hand, there is a belief that systems development should be considered to be a form of applied science or engineering (Dijkstra, 1976; Floyd, 1992; Hoare, 1982). This is also the approach behind Software Process Improvement. However, there are weaknesses with this approach which are identified below. These weaknesses are addressed by approaches from within the second paradigm or systems paradigm (Checkland, 1981; Checkland and Scholes, 1990). The systems paradigm has been developed from the traditional paradigm. Consequently, it is argued that fewer problems will be found if methodologies based on the systems paradigm are used. The foundations and nature of this approach are described. Further, it is shown that the Soft Systems Methodology (SSM), which is based on this paradigm, can be used as a metaphor or model of ISD. In this form it is recursive and so this model is characterised as Recursive SSM. SSM is offered as a useful framework for both ISD and software process improvement (SPI).

This paper will therefore address the following major activities:

- Examining the possible causes of and responses to the problem area recognised as the ‘software crisis’.
- Recognising that the above problems are not addressed by the science paradigm, per se. The science paradigm needs to be extended to address the outstanding issues. This extension is termed the systems paradigm.
- Identifying the support for the Soft Systems Methodology from the systems approach.
- Developing an understanding of the form of Recursive SSM.
- Recognising the use of SSM for ISD and SPI.
System Failures - Reasons and Responses

Software and its associated technologies and business processes are now transforming the opportunities for business organisations. Unfortunately, the promise of computer usage does not always bear fruit.

In 1968, a Study Group on Computer Science, established by the NATO Science Committee, responding to the perception of a “software crisis”, recommended the holding of a working conference on Software Engineering (Naur, Randell and Buxton, 1976). The term “software engineering” was coined to be deliberately provocative, implying, as it does, the need for software development to be based on the principles and practices seen in engineering (Naur et al., 1976).

While the introduction of an engineering approach had an effect on the way software is developed we note that some years after the initial concept of software engineering, Pressman (1987) still commented that for “... the past decade managers and many technical practitioners have asked the following questions:

• Why does it take so long to get programs finished?
• Why are costs so high?
• Why can’t we find all the errors before we give the software to our customers?
• Why do we have difficulty in measuring progress as software is being developed?”

Despite the problem represented by these questions being well known (in that these issues have been prominent in the literature for nearly thirty years), disasters are still happening.

While a common response to the software crisis has been to suggest that training should be improved (Canan, 1986; Hoare, 1982), practitioners and managers do not always do what they know they should. For example, Gibbs (1994) reports Larry E. Druffel, Director of Carnegie Mellon University’s Software Engineering Institute as saying that unfortunately, “... the industry does not uniformly apply that which is well-known best practice”. Even if “best practice” is enshrined in a methodology, this does not guarantee success. For example, a discussant (Eddie Moores) at the 1995 Information Systems Methodologies Conference, referring to some currently unpublished research of his, said that the most used methodology was “JDI” (Just Do It). It was also made clear in subsequent discussion that even when managers said a methodology was being used, this was no guarantee that this was so. Another example of managers not doing what is expected by the literature, is when they implement systems on time, but with known errors. As Pirsig (1984) notes, a quality product will only be produced if people care about quality.

The NATO Study Group’s belief that the use of applied science would be an appropriate solution to the software crisis was echoed by Hoare (1982) when he said that “professional practice ... [should be] ... based on a sound understanding of the underlying mathematical theories and ... should follow closely the traditions of engineers in better established disciplines.” Similarly, Dijkstra (1976) was interested in the mathematical basis of programming and considers the development of programs to be a scientific discipline. Gibbs (1994) still supports this view when he concludes that a disaster “… will become an increasingly common and disruptive part of software development unless programming takes on more of the characteristics of an engineering discipline rooted firmly in science and mathematics.” This is the same recipe as reported by Naur et al. (1976) nearly twenty years earlier. It is also the spur towards software quality assurance and SPI.

From the above description we can characterise the software engineering perspective as deterministic and one that assumes that there is some definable, true and real set of requirements that can be elicited and formally specified. It concentrates on the production of a piece of software that conforms to a specification as efficiently as possible (Vidgen, Wood-Harper and Wood, 1993). However, this optimistic approach lacks the insight shown by Bronowski (1973) when he says “… There is no absolute knowledge. And those who claim it, whether they are scientists or dogmatists, open the door to tragedy. All information is imperfect. ... That is the human condition ...”. While it might be overstating the issue to say an unsuccessful business problem is a tragedy, this paper will demonstrate that this insight does apply to ISD. The problem of unsuccessful information system developments is seen as unnecessary as the problems and solutions are well known.

Even when a system is installed, there can still be difficulties and disappointments. Little (1993), for example, finds estimates of system failure in the literature running between 25% and 90%. In a major survey of the literature on why information systems fail, Lyttinen and Hirschheim (1987) identify four generic issues which give rise to such failures:

• Correspondence failure - the system, as implemented, does not correspond to what was required;
• Process failure - a system is not forthcoming within time or resource constraints;
• Interaction failure - systems, as implemented, which fail to satisfy the users;
• Expectation failure - systems which are unable to meet stakeholders’ expectations.

The Process failure is clearly addressed (at least partly) by software engineering. However, the other three kinds of failures are not directly addressed by it. Recently, it has been recognised that it is important to recognise the way analysts are seen by the organisation and how they see themselves (Bell and Wood-Harper, 1992; Hirschheim and Klein, 1989). This can be identified by the use of metaphor. The software engineer is usually a technocrat who tries to deliver what is thought of as the user’s requirements, but with minimal reference to the user. The other failure types are more likely