Fitzgerald, B. (1994) The Systems Development Dilemma: Whether to Adopt Formalised Systems Development Methodologies or Not?, in Baets, W. (Ed) *Proceedings of Second European Conference on Information Systems*, Nijenrode University Press, Holland, pp. 691-706.

The Systems Development Dilemma: Whether to Adopt Formalised Systems Development Methodologies or Not?

Brian Fitzgerald
University College Cork, Ireland

Abstract

Systems development issues occupy a position of central importance in the information systems field and, indeed, much has been prescribed in the quest for successful systems development. However, given the well-documented "software crisis", success is far from guaranteed for many systems development projects. Many researchers see the solution to the software crisis in terms of increased control and the more widespread adoption of rigorous and formalised system development methodologies (SDMs), and this paper first presents the arguments and pressures in favour of formalised methodologies. However, the problems associated with the use of formalised methodologies have not perhaps received as much attention in the literature. A number of arguments are presented in this paper which question the value of formalised methodologies. These dichotomous arguments—for and against formalised SDMs—bring about a dilemma for systems developers as to whether to adopt a formalised development methodology or not. The implications of this dilemma are discussed in this paper and a number of issues for further research are proposed.

1. Introduction

The importance of successful systems development takes on even greater significance nowadays in view of the increasingly complex applications that need to be developed, and the well-documented problems associated with systems development which have given rise to what has been termed the "software crisis" (cf. Shemer, 1987). There are many researchers who see the solution to the software crisis in terms of increased control and more rigorous and formalised[1] system development methodologies (SDMs). A number of arguments can be made to support the use of these methodologies, and these arguments are presented in this paper. In addition, there are significant pressures on software developers to adopt more formalised SDMs, and these pressures are also discussed. However, the problems associated with the use of formalised methodologies have not perhaps received as much attention in the literature. A number of arguments are put forward which question the value of formalised SDMs. These opposing arguments represent a dichotomy which leaves systems developers facing a dilemma as to whether they should adopt more formalised SDMs or not. This issue is discussed and, finally, an outline of the research needed to resolve this dilemma is presented.

[1]The term 'formalised' is used here to denote rigorous, formally-defined development methodologies, of which there are many examples in the literature, rather than an ad-hoc approach to systems development, of which there are many examples in practice. Some writers use the term 'formal' in this context. However, this leads to confusion with those methodologies which have a mathematical basis for specification and design, which are also labelled as formal.

2. The Case for Formalised Methodologies

There are a very large number of SDMs available, estimated at more than 300 (Longworth, 1985). Much research has therefore focused on evaluating and comparing different SDMs—a very problematic task in itself. Table 1 provides a summary of this research. Several researchers seem to make the a priori assumption that the use of formalised SDMs is necessary and to be recommended (cf. Ramamoorthy et al, 1986; Yourdon, 1991). Indeed, some researchers have reported that in environments where less-formalised development methodologies are in use, the trend is to replace them with formalised development methodologies (Jenkins et al., 1984). A number of arguments may be made in favour of formalised SDMs. These arguments are

summarised in Table 2 and each is discussed in detail below.

2.1 Conceptual Basis for Methodologies

Early efforts at systems development often relied on unsystematic and random methods (Olerup, 1991, Yeh, 1991), although some systematic approaches to systems development were actually available (cf. Colter, 1984; Couger, 1973, Taggart and Tharp, 1977). An important early contribution, however, was that of Langefors (1973) who, in arguing for a more formal approach to system development, outlined the foundations for a theory of information systems. Langefors adopted a mechanistic view of organisations with optimal satisfaction of organisational goals as a central component, and such a view is evident in many current system development methodologies (Jones and Walsham, 1992). He conceptualised systems development as a rational and scientific process, and proposed a subdivision of the development process into deciding *what* an information system must do, and *how* it should do it (Langefors, 1973). Based on this rational scientific view, prevalent in many other disciplines, the development process is broken into the broad categories of analysis of requirements, design of a solution, and implementation of that solution (Olerup, 1991). Thus, operating from an ontological position of realism, systems development is conceptualised as a linear model, relying on phases ordered in time and leading to increasingly-formal defining documents until the software system eventually emerges (Floyd, 1987). This leads to the key concept of a system development life-cycle (SDLC) which contains as a central premise the subdivision of system development into several distinguishable sequential phases (Shemer, 1987), and which may be traced to the scientific reductionist mode of enquiry prevalent at the time (Agresti, 1986).

One of the widely-cited benefits of the phased approach to systems development is that it makes the development process more amenable to project management and control (Ahituv *et al*., 1984; Avison and Fitzgerald, 1988; Floyd, 1987; McCracken and Jackson, 1981; Ross and Brackett, 1976). At the end of each phase, there is an opportunity to review progress and to monitor actual costs and benefits and compare with expected figures, and this helps to minimise the risk inherent in systems development projects (McDonald *et al*., 1986). Also, since each phase is comprised of different tasks requiring different skills, some economics of specialisation are afforded (Olerup, 1991).

The economic theme is one mentioned by Baskerville *et al*, (1992) in discussing the rationale behind methods for systems development. They identify an economic rationale in so far as methodologies seek to eliminate irrational or counter-productive activities. Also, by providing a taxonomy of activities, development methodologies facilitate the grouping of similar activities and the reduction of redundant activities. Baskerville *et al* also propose an epistemological rationale for formalised development methodologies. This refers to the structural framework which methodologies provide, thereby allowing professionals working in the field to acquire and classify knowledge.

2.2 Pressures for Increased Formalism

There are a number of very influential sources which are causing an increased pressure in favour of the use of formalised development methodologies. For example, at a broad level the ISO-certification process, much sought after by organisations, requires the use of formalised development processes. Also, major institutions such as the UK government have mandated the use of the SSADM (Structured Systems Analysis and Design Method) methodology for system development. SSADM is now used on projects totalling billions of pounds each year (Downs *et al*., 1992), and this causes a significant pressure in the industry to move in this direction--a fact which is borne out in the large numbers of organisations supplying consultancy, training, and CASE tools supporting the SSADM methodology (Downs *et al*., 1992). Several other national governments have also adopted SSADM as the required development methodology, while countries such as France, Holland, and Italy have their own formalised development methodologies.

Similarly in the US, the Department of Defense (DoD) have established development standards (e.g. DoD Std. 2167) for software development which developers working on DoD projects must follow. These standards have emerged from several years' research and are intended to allow the DoD more visibility and control with respect to the development process (Coad and Yourdon, 1991). Also, the DoD have recently collaborated with the Software Engineering Institute on the Software Capability Evaluation (SCE) programme. This programme is concerned with assessing the capability of organisations to produce quality software in a timely and repeatable fashion and it has generated

intense focus in the US software industry (Bollinger and McGowan, 1991). However, this programme places great emphasis on adherence to formalised development procedures. Indeed, its advocates suggest that effective development requires that all steps in a development methodology should be carried out regardless of circumstances (Humphrey *et al*., 1991). This is a controversial issue as it fails to take account of contingencies of any particular situation--a factor discussed in the next section. This emphasis on formalised approaches is also consistent with the classical stage approach to computer growth and management as proposed by Gibson and Nolan (1974) which suggests that organisations adopt more formalised approaches to managing software processes as they mature.

There is also a great deal of interest now in formal mathematically-based methods, such as VDM, Gist, PAISLey, Z (cf. e.g. Balzer *et al*., 1982; Prehn & Toetenel, 1991; Zave, 1984), as a basis for systems development. These methods facilitate automatic transformation from requirements specification to the final system, and are suggested to be capable of producing higher quality software at a lower cost than with conventional methods (Plat *et al*., 1991). Formal methods have a mathematical basis and allow rigorous validation and verification of designs during the development process (Plat *et al*., 1991). This is in contrast to more traditional development methods which are purely descriptive and rely on textual descriptions, and which consequently are prone to imprecision and ambiguity (Alexander and Potter, 1987; Docker, 1987). Formal methods are suggested to be necessary for effective software development (Docker, 1987), and researchers have reported growing interest by industry in the use of formal methods (Wing and Zaremski, 1991).

3. The Case against Formalised Methodologies

The assumption that formalised development methodologies actually represent the most appropriate means of solving the software crisis is open to question. More than 300 different system development methodologies have been identified (Longworth, 1985), and at least 17 different systems development life-cycle variations have been proposed (Necco *et al.* 1987). But the question has been posed as to whether there are actually substantially different ways to develop systems (Olle *et al*. 1991). Systems development methodologies are attractive and have an intuitive appeal, but a systems development methodology is not system development, rather it is a framework for organising the system development process. Indeed, a large part of the methodology may go towards justifying the methodology itself (Andersen & Mathiassen, 1987). Some of the problem areas for formalised development methodologies are summarised in Table 3 and each is discussed in detail below.

3.1 Definitional problems

When it comes to deciding what actually constitutes a development methodology, the definitional quagmire so common in the computing field becomes apparent. The term methodology is often misused in that the term actually means 'study of method' (Olle *et al*., 1991). However, methodologies have been variously defined in terms of models, management practices, technical practices, tools, training procedures and so on (De Grace & Stahl, 1990). This theme is echoed by Maddison *et al* (1984) who acknowledge the problem of identifying what actually constitutes a methodology, and propose a broad inclusive definition of a development methodology as "a recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management and training for developers of information systems".

Given the large number of methodologies available, some have suggested that there may not be significant differences between different methodologies. For example, Constantine, one of the founding figures of the structured approach, admits that different development approaches are actually "based on product differentiation, personal ego, and territorial imperative" (Constantine, 1989). This view is supported by Veryard (1985) who suggests that there are trivial differences between many methodologies. Indeed, it is argued that the 'software crisis' has been grossly exaggerated to rationalise new development approaches in the software development arena (DeGrace and Stahl, 1990). However, other researchers point to the fundamental differences between methodologies in terms of philosophy, objective and techniques (Avison & Fitzgerald, 1988). Methodologies may differ fundamentally in paradigm--from 'hard' scientific to 'soft' human-oriented, and in focus, as some methodologies do not cover requirements analysis while others do not cover implementation (Sakthivel, 1992).

Methodologies have often been constructed by abstracting some features and techniques from successful development projects, and formalising these into a set of guidelines and procedures to

form a development methodology, but there may be little philosophical justification (Maddison *et al*., 1984). For example, the structured approach to systems development, the "most popular systems development methodology in North America and Europe" (Yourdon, 1991), was based on the intuition of its founders that it would work rather than on any real-world experience (Ward, 1992a). According to one of the founders, early investigations of the structured approach were just "noon hour critiques" (Yourdon & Constantine, 1977). In fact, several areas of weakness in the structured approach have been identified (cf. Coad and Yourdon, 1991; Henson and Hughes, 1991; McMenamin and Palmer, 1984; Ward, 1992a).

3.2 Inadequacies of the Scientific Paradigm

The underlying paradigm for many development methodologies is the scientific reductionist one (Baskerville *et al*, 1992; Wood-Harper and Fitzgerald, 1982). There is an *a priori* assumption that the solution can be arrived at through a series of technically devised steps (Davies and Ledington, 1991; Hackathorn and Karimi, 1988) and that the developer can obtain detailed knowledge about the problem situation (Giddings, 1984). The latter is questioned by Jones and Walsham (1992) who suggest there are "limits to the knowable". They argue that it is not possible, nor indeed appropriate in all cases, for the designer to obtain detailed knowledge about the organisation and design context. Boland (1979) also questions the extent to which an organisational problem exists as an independent reality that can be modelled beforehand, and he identifies the critical importance of how the situation is interpreted by the actors in the situation.

Other researchers have also questioned the validity of viewing systems development as a rational process. For example, Robey and Markus (1984) argue that while the various phases of systems development can be explained by rational motives, they can also be explained as political rituals which are used to negotiate the private interests of the various parties concerned. Thus, the stages in systems development can be explained by two diametrically-opposed sets of motives--rational and political. However, the rational motives are the ones assumed in many traditional systems development methodologies, and consequently, they do not cope well with social and human factors (Bostrom and Heinem, 1977; Floyd, 1987; Goldkuhl and Lyytinen, 1984). System development is not just a technical process; social needs to be considered also (Baskerville *et al*, 1992; Land *et al*, 1980). Yet, researchers have suggested that most development methodologies only pay lip-service to social aspects and have argued that treating system development as a purely-technical process is a "recipe for disaster" (Hirschheim and Newman, 1991). Consequently, much research has focused on softer approaches to systems development which counter these criticisms (cf. e.g. Checkland, 1984; Floyd, 1987; Hirschheim and Newman, 1991; Land *et al*., 1980; Mumford, 1984; Olerup, 1991).

Parnas and Clements (1986) also argue that the rational approach to systems development that is part of many development methodologies is not valid, as it is a much less tidy process in practice. However, they suggest that there are good reasons for performing some purification on the results of systems development to "fake" a rational approach to development. Anyone who must work on the system after it is developed will want to understand it, not relive its discovery, and this is best achieved by access to the polished output of what can justifiably masquerade as having been a rational process.

3.3 Shortcomings of the Waterfall Life-Cycle framework

Most development methodologies follow the waterfall lifecycle (Davis *et al*., 1988; Orr, 1989). The waterfall life-cycle is logical and appeals to management. It recognises the importance of analysis and design, rather than rushing to the program coding phase. In an era when computers were very expensive and outnumbered by programmers (Musa, 1983), an initial analytical phase was logical and sensible. However, there is an imbalance between analysis and synthesis in the waterfall life-cycle (Agresti, 1986). Agresti says that the field has evolved to the point where synthesis should be more widely used in software development, and he points to other engineering disciplines where there is a "rich interplay" between analysis and synthesis.

A phased approach, such as that underpinning the waterfall life-cycle, is actually quite a common approach to problem-solving in many disciplines. However, the main problem with the life-cycle approach in the systems development area has to do with the rigid and inflexible manner in which it has generally been applied. Glass (1991) criticises it as an "ironclad set of rules...an inviolate approach that has to be followed in just the right order and just the right way". McCracken and Jackson (1981) make a similar point. They see the waterfall life-

cycle as a project management approach imposed on the system development process. They criticise the life-cycle process because it rigidifies thinking, and they describe the completion of individual phases in the waterfall approach as leading to a "sawtooth" model of system development. Several other researchers have criticised the phase dependencies implicit in the waterfall life-cycle (Henson & Hughes, 1991; Parnas & Clements, 1986; Shemer, 1987; Swartout & Balzer, 1982, Vitalari & Dixon, 1983). In real-life systems development, there is an inevitable intertwining of specification and implementation since problems are dynamic and actually change as they are being solved. Also, there are wide variations in the number and labelling of phases in the waterfall lifecycle (Necco *et al*., 1987), and it has been criticised because the granularity of individual phase steps is too large, thus failing to show all the elemental processes within each phase (Curtis *et al*., 1992).

A fundamental assumption of the waterfall life-cycle is that of proceeding from an initial stage of requirements analysis which are then frozen, through to solution design and implementation (Land, Mumford & Hawgood, 1980). However, there are several flaws inherent in such an approach. Firstly, modern organisations are characterised by rapid change--there is no "organisational stasis" (Chikofsky, 1989). Therefore, it is not appropriate to consider a fixed organisational framework when developing systems as some methodologies do. Brooks (1987) also contends that the assumption of the waterfall life-cycle that requirements can be specified in advance is fundamentally wrong. He states that the hardest part of systems development is determining the specification of what to develop. Parnas and Clements (1986) are in agreement, pointing out that users typically do not know their complete requirements, and that there is inevitable backtracking as development takes place. Davis *et al.* (1988) describe user needs as a "moving target" which are "constantly evolving", and it is therefore inappropriate to try freeze requirements in the specification phase. This attempt to finalise requirements before any development takes place does not occur in other disciplines (Glass, 1991). McCracken and Jackson (1981) consider the situation to be analogous to deciding all item purchases upon entry to a supermarket. They suggest that the waterfall life-cycle may have seemed appropriate in the past due to the complexity of system development, but it now perpetuates the failure to bridge the communication gap between user and analyst.

Systems development in practice is not an orderly systematic phased process, rather it happens "all at once" (DeGrace & Stahl, 1993). In the waterfall life-cycle the *what* of requirements specification is strictly separated from the *how* of design. Yet, as Peters (1981) bluntly puts it: "one cannot state a problem without some rudimentary notion of what the solution should be". Shemer (1987) suggests a jigsaw analogy. He argues that the ultimate design is not achieved in a rational top-down manner. Rather, information is obtained in random order; some design is done bottom-up by guessing at a local pattern, and, simultaneously, some design is done top-down by following an apparent pattern. A study by Zelkowitz (1988) lends validity to this, reporting that 50 percent of design activity occurs in phases other than the design phase of the waterfall life-cycle.

Other approaches to systems development, such as evolutionary development and prototyping, have emerged in response to some of the inadequacies of the waterfall life-cycle, particularly the suggestion that requirements can be specified in advance. These approaches are characterised by the evolutionary nature with which the system is produced in an iterative fashion, perhaps through a series of prototypes (cf. e.g. Agresti, 1986; Davis *et al*., 1988; Mayhew and Dearnley, 1987; Mansuy, 1989).

3.4 Goal Displacement

One of the most harmful implications that may arise through the use of a development methodology is that of goal displacement. This refers to the situation whereby developers become preoccupied with slavish adherence to the methodology at the expense of actual development; that is, the developer becomes engrossed in following the methodology and loses sight of the fact that development of a system is the real goal (DeGrace & Stahl, 1990). Further compounding the problem is the fact that many methodologies include logically-redundant tasks so as to improve reliability, but developers often perform unnecessary tasks and omit necessary ones (Veryard, 1985).

As discussed above, part of the rationale behind the use of development methodologies is to facilitate project management and control of the development process, and methodologies have an intuitive appeal for management. However, Glass (1991) compares the use of a development methodology to the effect of the Maginot line--giving the "illusion of quality but hiding violations". Development methodologies attempt to

impose complete solutions when the minimum are not yet well-defined. A fundamental problem arises when the methodology is treated in a catechistic fashion, as this may give rise to an inflexible approach in which it becomes difficult to take advantage of opportunities or deal with contingencies. Glass suggests that the software field is too young for "premature positions and posturings". He argues that methodologies focus on the trappings of design rather than on its essence which is actually the cognitive activity in the mind of the developer.

3.5 Assumption that Methodologies are Universally Applicable

Also, methodologies are often promoted as the "one-best way" which leads to an elaborate and bureaucratic" approach to systems development (Benyon & Skidmore, 1987). There may be a tendency to blindly follow a development methodology on the assumption that it is universally applicable in all situations (Giddings, 1984). This does not give due consideration to the contingencies of each development situation, since the developer creates a unique situation for every project (Avison *et al*., 1988; Curtis *et al.,* 1988). In practice, developers frequently do not apply the methodologies in their complete form as specified (Chikofsky, 1989, Jenkins *et al*., 1984). Developers omit those aspects of the methodology that do not seem to suit the contingencies of the situation. For example, the US Department of Defense, whose strong advocacy of formalised methodologies has already been discussed, recommends tailoring of methodologies to suit the particular development situation (Chikofsky, 1989; Coad and Yourdon, 1991; DeGrace and Stahl, 1990).

Many other researchers reject the notion of a slavish and rigid adherence to a development methodology: Baskerville *et al.* say that software development in practice is actually an unstructured evolutionary process, and they suggest that methodologies can be a "burden" and a "destructive tyrant" for the developer. Studies of system development show that chaos is endemic and "things happen all at once" (DeGrace & Stahl, 1993). In many instances, however, development methodologies are inflicted on developers rather than made available, and a rigid dogmatic approach to development is taken. However, there is a need to be able to step outside the methodology to take advantage of opportunities or to deal with exigencies that may arise. Also, an interesting finding emerges from a study by DeMarco and Lister (1989) which shows that even in organisations where methodologies are rigidly enforced, there is very poor convergence on design style among different developers. This again reinforces the point that development methodologies cannot be inflicted on developers.

3.6 Inadequate Recognition of People Factors

Boland (1979) argues that organisational problem situations do not exist as an independent reality but require human interpretation, a point also raised by Davies & Ledington (1991). This is dependent on the people involved and as Checkland (1984) points out, "uniformity of perception cannot be imposed on autonomous human beings". The ingenuity and ability of the developer cannot be compounded into any development methodology. At a simplistic level, considering an analogy between cookbooks and development methodologies, no one believes that merely having access to the same cookbook would cause all chefs to be equally proficient. However, the varied skill levels of different developers is not acknowledged in formalised development methodologies. For example, one of the explicit goals of the Jackson Systems Development (JSD) methodology is to eliminate personal creativity from the development process (King and Pardoe, 1985). Yet, Brooks (1987) suggests that systems development is a creative process and that a methodology cannot "inspire the drudge". The importance of individual differences in system development has been acknowledged by several researchers. Boehm (1981) reports that the people factors have more than six times greater effect on development productivity than the use of software tools. Brooks (1987) is in accord with this view and he recommends that processes be put in place to nourish creative people. He states that few fields have such a large gap between best current practice and average practice. Indeed, Glass (1991) reports differences of up to 30 to 1 between software developers.

Nor do methodologies allow for the learning experience and greater problem domain knowledge that developers gain over time. Yet, in a comparative study of successful and unsuccessful systems analysts, Vitalari and Dickson (1983) emphasise the importance of learning over time. They conclude that developers acquire a "repertoire of strategies" to apply in different system development situations. This is in accord with Davis and Olson (1985) who suggest that developers gain more domain knowledge over time and that this is a vital factor in successful system

development. To view system development as an orderly progression from requirements analysis to a solution designed purely around those requirements is to miss the critical synergy between developer and user. Curtis *et al.* (1988) have suggested that both the developer and user learn through a dialectic approach, in that by hearing about potential capabilities of the system, users envision new features (Swartout & Balzer, 1982). Vitalari and Dickson (1983) report that successful designers learn a great deal through trial and error. Therefore, an idealised approach to system development as portrayed in a methodology may be seriously flawed since it omits the fact that failure is essential to human learning.

4. Whither Systems Development?

While the rationale behind the use of formalised development methodologies is persuasive, the arguments against the use of formalised methodologies are also compelling. In practice, however, many practitioners do not use a formalised system development methodology (Avison & Fitzgerald, 1988; Page-Jones, 1991; Ward, 1992a). A number of reasons have been put forward to explain this. For example, it has been suggested that the failure to use a formalised methodology is due to a "wealth of ignorance" among the "great unwashed masses" (Ward, 1991), and the failure of practitioners to use development methodologies is seen as a weakness on their part (Page-Jones, 1991). Also, it has been suggested that it takes about 15-20 years for technology transfer to achieve sufficient maturity for general use (Chikofsky, 1989), and that this is what has delayed the adoption of methodologies. However, in all this research, there is an implicit assumption that the failure of practitioners to use formalised methodologies has been to the detriment of systems development.

Non-use of a methodology is not a licence to conduct development in a sloppy or careless manner. Those who suggest that the failure of practitioners to use a formalised methodology is due to ignorance or a lack of awareness on their part may not be presenting a totally-accurate picture. An appropriate analogy might be that of Picasso dispensing with conventional artistic perspective, but from a position of superior knowledge. Likewise, many practitioners may be well aware of the limitations of formalised methodologies and may have rejected them for pragmatic reasons. As a practitioner with over 10 years experience of systems development in different organisations on many different applications, the author has yet to witness a development project where a formalised development methodology was faithfully adhered to. In practice, situations will inevitable arise where the developer needs to step outside the methodology, but formalised methodologies often serve to impose a considerable inertia on the development process. Indeed, the degree of inertia is proportional to the degree of formality of the methodology.

A number of factors were discussed earlier as being very important to the success of the system development process. These include the critical differences in capabilities between developers; the importance of learning over time, both in terms of ensuring increased problem domain knowledge and also the exposure to a variety of technical problem-solving strategies. The ability to use intuition in an appropriate manner is also an important asset for systems developers. However, these are all factors which are not adequately catered for in formalised development methodologies. Systems development is not just about knowing the phases and activities involved in a development methodology, rather the developer should comprehend the underlying concepts. Development methodologies are just an organising framework, and are only meaningful when applied by people. It is important, therefore, that a methodology fully leverages the wisdom of the developer, arising both from individual ability and past learning experiences, if it is to make the most effective contribution to the development process.

4.1 Pressures for New Approaches to Systems Development

There are a number of pressures for new and radical approaches to systems development which do not support the use of formalised development methodologies. The accelerating pace of change characteristic of the business environment facing organisations today is a common theme in contemporary research. Rockart and De Long (1988) refer to the "faster metabolism of business today" which requires organisations to act effectively in shorter time-frames. Researchers have estimated a need for a ten-fold increase in system development productivity (Verity, 1987), but formalised methodologies for systems development are oriented towards large-scale development with a long development time. Given the continuous change that organisations are now faced with, short-term needs dominate, and these in turn mean that the economics of formalised systems development is dwindling (Baskerville *et*

al., 1992). Developers do not have the luxury of being able to patiently follow a comprehensive methodology. Indeed, the truncation of some phases in the development process is seen as inevitable (Brown, 1985).

In many disciplines there is a natural progression of improving the process by which products are produced. The situation is no different in the software field. Development methodologies are becoming more complex; for example, methodologies such as Information Engineering, SSADM, and Multiview are very comprehensive and address a broad range of phases involved in software development. These methodologies are evolving as new concerns and areas of focus emerge in the field. This type of evolution is consistent with the views of philosophers of science, such as Lakatos (1970) who argues that as disciplines progress, they erect a protective belt of sub-theories around core theories to cater for problem areas and criticisms. However, Kuhn (1962) suggests that progress in science requires that established paradigms are eventually overthrown and replaced, often with conceptually-cleaner paradigms. In the software field, researchers have suggested that improving the process by which software is developed can only have a limited effect, and that a software industrial revolution which focuses on radically new ways to achieve the software product is necessary, and, consequently, new paradigms for systems development have been advocated (cf. Agresti, 1986; Cox, 1990).

4.2 Further Research

Boehm (1988) has criticised the focus of research in the software field as being directed towards certain well-understood areas while neglecting other areas which are less well-understood but equally important. He proposes the analogy of a drunk losing his watch and looking for it under the light of a lamppost because it was the brightest place even though he had lost it somewhere else. The situation may be similar with the excessive focus on formalised methodologies, in that it may be case of looking under the lamppost. Certainly, Lewis and Oman (1990) claim that in 20 years, systems development has "evolved to little more than a black art", and Wasserman's contention (1981) that the greatest boost to systems development productivity would be to teach programmers the skills of touch-typing has yet to be refuted.

Researchers have criticised the lack of empirical research on systems development in real organisational contexts (Jenkins *et al*., 1984). As McLean (1973) aptly put it: "the proper place to study elephants is the jungle, not the zoo". More research is therefore needed into the actual practice of systems development in organisations, justifiable even solely on the basis that practice has often preceded theory in the field. Programming style, compiler writing, user-interface design are all areas where practice led theory (Glass, 1991). The Sage missile-defense system and the Sabre airline reservation system, developed in the 1950s and 1960s, were both examples of sophisticated interactive systems which far exceeded the maturity of the theory at the time (Shaw, 1990). Also, given the wide gap between the best and average practice in the software field (cf. Boehm, 1981; Brooks, 1987; Glass; 1991), it is important to discover the essentially good practices of good systems developers, so that these can be transferred to other developers. All too often, however, theorists fail to consider practice when it might be appropriate to do so, and, vice versa, practitioners fail to heed theorists when it might be beneficial. Researchers have criticised the gap between theory and practice, whereby theorist and practitioners are isolated from each other and move in different directions, labelling it a "bipolar drift (in which) both poles are cold" (Chang, 1990).

This paper is preliminary in nature, presenting as it does the dichotomous arguments which bring about a dilemma for system developers as to whether to adopt a formalised SDM or not. The next stage of this research will be to empirically examine the issues raised in this paper. Among the specific research questions to be answered are the extent to which formalised methodologies are actually used; whether they are followed faithfully or modified; in the cases where formalised methodologies are not used, whether this is due to ignorance, or for more pragmatic reasons; the benefits that accrue from the use of formalised methodologies; the situations where developers would consider using or not using a methodology.

By addressing these issues, the research should thus help to ascertain whether practitioners are indeed moving towards more formalised development methodologies as has been suggested and, indeed, recommended; or whether there is a sense in which software development is perhaps beyond method in some circumstances. Given the dichotomous nature of the arguments posed in this paper, it is perhaps worth bearing in mind Niels Bohr's reminder that the opposite of a great truth is also true. In other words, while for many researchers the use of formalised development methodologies is unquestionably beneficial and

represents a great truth, the opposite view, namely, that systems development without formalised development methodologies may also be appropriate, is no less a truth.

Bibliography

Agresti, W. (1986) New Paradigms for Software Development. IEEE Computer Society Press, Washington DC.

Ahituv, N., Hadass, M. and Neumann, S. (1984) A flexible approach to information system development. *MIS Quarterly*, **June**, 69-78.

Alexander, H. and Potter, B. (1987) Case study: the use of formal specification and rapid prototyping to establish product feasibility. *Information & Software Technology*, **September**, 388-393.

Andersen, P. and Mathiassen, L. (1987) Systems development and use: a science of truth or a theory of lies. In *Computers and Democracy: A Scandinavian Challenge*. Bjerknes, G., Ehn, P., and King, M. (Eds), Avebury Gower, Brookfield Vermont.

Avison, D. and Fitzgerald, G. (1988) *Information Systems Development: Methodologies, Techniques and Tools*. Blackwell Scientific Publications, Oxford.

Avison, D., Fitzgerald, G. and Wood-Harper, A. (1988) Information systems development: a tool kit is not enough. *The Computer Journal*, **31**, 4, 379-380.

Balzer, R., Goldman, N. and Wile, D. (1982) Operational specification as the basis for rapid prototyping. ACM Sigsoft *Software Engineering Notes*, **7**, 5, 3-16.

Banbury, J. (1987) Towards a framework for systems analysis practice. In Boland, R and Hirschheim, R. (eds.) *Critical Issues in Information Systems Research*, John Wiley and Sons, 79-111.

Bantleman, J. and Jones, A. (1984) Systems analysis methodologies: a research project. In Bemelmans, T. (ed.) *Beyond Productivity: Information Systems Development for Organisational Effectiveness*, Elsevier Science Publishers B.V., North Holland Press, 213-227.

Baskerville, R., Travis, J. and Truex, D. (1992) Systems without method: the impact of new technologies on information systems development projects. In Kendall, K., DeGross, J. and Lyytinen, K. (eds.) *The Impact of Computer Supported Technologies on Information Systems Development*, Elsevier Science Publishers B.V., North Holland Press, 241-269.

Benyon, D. and Skidmore, S. (1987) Towards a toolkit for the systems analyst. *The Computer Journal*, **30**, 1, 2-7.

Boehm, B. (1981) *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, New Jersey.

Boehm, b. (1988) In Gilb, T. *Principles of Software Engineering Management*, Addison Wesley, UK.

Boland, R. (1979) Control, causality and information systems requirements. *Accounting, Organizations and Society*, **4**, 259-275.

Bollinger, T. and McGowan, C. (1991) A critical look at software capability evaluations. *IEEE Software*, **July**, 25-41.

Bostrom, R. and Heinem, J. (1977) MIS problems and failures: a socio-technical perspective--Part I: the causes. *MIS Quarterly*, **September**, 17-32.

Brooks, F. (1987) No silver bullet: essence and accidents of software engineering. *IEEE Computer Magazine*, **April**, 10-19.

Brown, P. (1985) Managing software development. *Datamation*, April 15, 133-136.

Chang, C. (1990) Editor's message: let's stop the bipolar drift. *IEEE Software*, **May**, 4.

Checkland, P. (1981) *Systems Thinking, Systems Practice*, Wiley, Chichester.

Checkland, P. (1984) Systems theory and information systems. In Bemelmans, T. (ed.) *Beyond Productivity: Information Systems Development for Organisational Effectiveness*, Elsevier Science Publishers B.V., North Holland Press, 9-21.

Chikofsky, E. (1989) How to lose productivity with productivity tools. *Proceedings of 3rd IFAC/IFIP Workshop*, Indiana, US, 1-4.

Coad, P. and Yourdon, E. (1991) *Object-Oriented Analysis*, (2nd Edition), Yourdon Press, New Jersey.

Colter, M. (1984) A comparative examination of systems analysis techniques. *MIS Quarterly*, **March**, 51-66.

Constantine, L. (1989) The structured design approach. *Byte*, **April**, 232-233.

Couger, J. (1973) Evolution of business systems analysis techniques. *Computing Surveys*, **5**, 3, 167-198.

Cox, B. (1990) Planning the software industrial revolution. *IEEE Software*, **November**, 25-33.

Curtis, B., Kellner, M. and Over, J. (1992) Process Modelling. *Communications of the ACM*, **September**, 75-90.

Curtis, B., Krasner, H. and Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, **November**, 1268-1287.

Davies, L. and Ledington, P. (1991) *Information in Action: Soft Systems Methodology*. Macmillan Press, London.

Davis, A., Bersoff, E. and Comer, E. (1988) A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, **October**, 1453-1460.

Davis, G. (1982) Strategies for information requirements determination. *IBM Systems Journal*, **21**, 1, 4-30.

Davis, G. and Olson, M. (1985) *Management Information Systems: Conceptual Foundations, Structure and Development*, McGraw-Hill, New York.

De Grace, P and Stahl, L. (1990) *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms*. Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey.

DeGrace, P. and Stahl, L. (1993) *The Olduvai Imperative: CASE and the State of Software Engineering Practice*, Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey.

De Marco, T. and Lister T. (1989) Software development: state of the art v. state of the practice. *11th International Conference on Software Engineering*, 271-275.

Docker, T. (1987) A flexible software analysis tool. *Information & Software Technology*, **January/February**, 21-26.

Downs, E., Clare, P. and Coe, I. (1992) *Structured Systems Analysis and Design Method: Application and Context*. Prentice-Hall International(UK), Hertfordshire.

Floyd, C. (1987) Outline of a paradigm change in software engineering. In *Computers and Democracy: A Scandinavian Challenge*. Bjerknes, G., Ehn, P., and King, M. (Eds), Avebury Gower, Brookfield Vermont.

Gibson, C. and Nolan, R. (1974) Managing the four stages of EDP growth. *Harvard Business Review*, **52**, 76-88.

Giddings, R. (1984) Accommodating uncertainty in software design. *Communications of the ACM*, **May**, 428-434.

Glass, R. (1991) *Software Conflict: Essays on the Art and Science of Software Engineering*. Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey.

Goldkuhl, G. and Lyytinen, K. Information systems specification as rule construction. In Kendall, K., DeGross, J. and Kyytinen, K. (eds.)*Beyond Productivity: Information Systems Development for Organisational Effectiveness*, Elsevier Science Publishers B.V., North Holland Press, 79-94.

Gould, J. and Lewis, C. (1985) Designing for usability: key principles and what designers think. *Communications of the ACM*, **March**, 300-311.

Gremillion, L. and Pyburn, P. (1983) Breaking the systems development bottleneck. *MIS Quarterly*, **March/April**, 130-137.

Guimares, T. (1985) A study of application program development techniques. *Communications of the ACM*, **May**, 494-499.

Hackathorn, R. and Karimi, J. (1988) A framework for comparing information engineering methods. *MIS Quarterly*, **June**, 202-220.

Henson, K. and Hughes, C. (1991) A two-dimensional approach to systems development. *Journal of Information Systems Management*, **Winter**, 35-43.

Hirschheim, R. (1985) User experience with and assessment of participative systems design. *MIS Quarterly*, **December**, 295-304.

Hirschheim, R. and Newman, M. (1991) Symbolism and information systems development: myth, metaphor and magic. *Information Systems Research*, **2**, 1, 29-62.

Humphrey, W., Snyder, T. and Willis, R. (1991) Software process improvement at Hughes Aircraft. *IEEE Software*, **July**, 11-23.

Jenkins, A., Naumann, J. and Wetherbe, J. (1984) Empirical investigation of systems development practices and results. *Information & Management*, **7**, 73-82.

Jones, M. and Walsham, G. (1992) The limits of the knowable: organizational and design knowledge in system development. In Kendall, K., DeGross, J. and Lyytinen, K. (eds.) *The Impact of Computer Supported Technologies on Information Systems Development*, Elsevier Science Publishers B.V., North Holland Press, 195-213.

King, M. and Pardoe, J. (1985) *Program Design using JSP: A Practical Introduction*, Macmillan, London.

Kuhn, T. (1962) *The Structure of Scientific Revolutions*, University of Chicago Press, Chicago.

Lakatos, I. (1970) *Criticisms of the Growth of Knowledge*, Cambridge University Press, Cambridge.

Land, F., Mumford, E. and Hawgood, J. (1980) Training the systems analyst of the 1980s: four analytical procedures to assist the design process. In Lucas, H., Land, F., Lincoln, and Supper (eds.) *The Information Systems Environment*, North Holland Press, 239-256.

Langefors, B. (1973) *Theoretical Analysis of Information Systems*, Auerbach, Philadelphia.

Lewis, T. and Oman, P. (1990) The challenge of software development. *IEEE Software*, **November**, 9-12.

Longworth, G. (1985) *Designing Systems for Change*. NCC, Manchester.

McCracken, D. and Jackson, M. (1981) A minority dissenting position. In Agresti, W. (1986) *New Paradigms for Software Development*. IEEE Computer Society Press, Washington DC.

McDonald, C., Riddle, W. and Youngblut, C. (1986) STARS methodology area summary. *ACM Software Engineering Notes*, **11**, 2, 58-85.

McLean, E. (1973) In Van Horn, R. Empirical studies of management information systems. *DataBase*, **Winter**, 172-180.

McMenamin, S. and Palmer, J. (1984) *Essential Systems Analysis*, Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey.

Maddison, R., Baker, G., Bhabuta, L., Fitzgerald, G., Hindle, K., Song, J.,Stokes, N. and Wood, J. (1984) Feature analysis of five information system methodologies. In Bemelmans, T. (ed.) *Beyond Productivity: Information Systems Development for Organisational Effectiveness*, Elsevier Science Publishers B.V., North Holland Press, 277-306.

Mahmood, M. (1987) System development methods- a comparative investigation. *MIS Quarterly*, **September**, 293-311.

Mansuy, J. (1989) Evolutionary development strategy for MIS. *Journal of Systems Management*, **July**, 7-13.

Martin, J. and Finkelstein, C. (1981) *Information Engineering*, Savant Institute, UK.

Mayhew, P. and Dearnley, P. (1987) An alternative prototyping classification. *Computer Journal*, **30**, 6.

Mumford, E. (1984) Participation- from Aristotle to today. In Bemelmans, T. (ed.) *Beyond Productivity: Information Systems Development for Organisational Effectiveness*, Elsevier Science Publishers B.V., North Holland Press, 95-104.

Musa, J. (1983) Stimulating software engineering progress. *ACM Software Engineering Notes*, **8**, 2, 29-54.

Necco, C., Gordon, C. and Tsai, N. (1987) Systems analysis and design: current practices. *MIS Quarterly*, **December**, 1987.

Olerup, A. (1991) Design approaches: a comparative study of information system design and architectural design. *The Computer Journal*, **34**, 3, 215-224.

Olle, T., Sol, H. and Verrijn-Stuart, A. (1982) *Information Systems Design Methodologies: A Comparative Review*, North-Holland.

Olle, T., Sol, H. and Tully, C. (1983) *Information Systems Design Methodologies: A Feature Analysis*, North-Holland.

Olle, T., Sol, H. and Verrijn-Stuart, A. (1986) *Information Systems Design Methodologies: Improving the Practice*, North-Holland.

Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Van Assche, F. and Verrijn-Stuart, A. (1991) *Information Systems Methodologies: A Framework for Understanding*. Addison-Wesley.

Orr, K. (1989) Methodology: the experts speak. *BYTE*, **April**, 221-233.

Page-Jones, M. (1991). Structured methods are dead: long live structured methods. *American Programmer*, **November**, 31-37.

Parnas, D. and Clements, P. (1986) A rational design process: how and why to fake it. *IEEE Transactions on Software Engineering*, **February**, 251-257.

Peters, L. (1981) *Software Design: Methods and Techniques*. Yourdon Press, New York.

Peters, L. and Tripp, L. (1977) Comparing software design methodologies. *Datamation*, **November**, 89-94.

Plat, N., Katwijk, J. and Pronk, K. (1991) A case for structured analysis/formal design. In *VDM '91: Formal Software Development Methods*, Prehn, S. and Toetenel, W. (Eds), Vol. 1, Springer-Verlag.

Prehn, S. and Toetenel, W. (1991) *VDM '91: Formal Software Development Methods*, Vol. 1, Springer-Verlag.

Ramamoorthy, C., Garg, V. and Prakash, A. (1986) Programming in the large. *IEEE Transactions on Software Engineering*, **July**, 769-783.

Robey, D. and Markus, M. (1984) Rituals in information system design. *MIS Quarterly*, **March**, 5-15.

Rockart, J. and De Long, D. (1988) *Executive Support Systems*, Dow Jones-Irwin, Homewood, Illinois.

Ross, D. and Brackett, J. (1976) An approach to structured analysis. *Computer Decisions*, **September**, 40-44.

Sakthivel, S. (1992) Methodological requirements for information systems development. *Journal of Information Technology*, **7**, 141-148.

Shaw, M. (1990) Prospects for an engineering discipline of software. *IEEE Software*, **November**, 15-24.

Shemer, I. (1987) Systems analysis: a systematic analysis of a conceptual model. *Communications of the ACM*, **June**, 506-512.

Soloway, E., Littman, D., Soloway, E., and Black, J. (1983) You can observe a lot by just watching (how designers design). *Eighth Annual Software Engineering Workshop*.

Song, X. and Osterweil, L. (1992) Toward objective, systematic design-method comparisons. *IEEE Software*, **May**, 43-53.

Sumner, M. and Sitek, J. (1986) Are structured methods for systems analysis and design being used? *Journal of Systems Management*, **June**, 18-23.

Swartout, W. and Balzer, R. (1982) On the inevitable intertwining of specification and implementation. *Communications of the ACM*, **July**, 438-440.

Taggart, W. and Tharp, M. (1977) A survey of information requirements analysis techniques. *Computing Surveys*, **9**, 4, 273-290.

Verity, J. (1987) The OOPS revolution. *Datamation*, May 1, 73-78.

Veryard, R. (1985) What are methodologies good for? *Data Processing*, **July/August**, 9-12.

Vitalari, N. and Dickson, G. (1983) Problem solving for effective systems analysis: an experimental exploration. *Communications of the ACM*, **November**, 948-956.

Ward, P. (1991) The evolution of structured analysis: Part I--the early years. *American Programmer*, **4**, 11, 4-16.

Ward, P. (1992a) The evolution of structured analysis: Part II--maturity and its problems. *American Programmer*, **5**, 4, 18-29.

Ward, P. (1992b) The evolution of structured analysis: Part III--spin-offs, mergers, and acquisitions. *American Programmer*, **5**, 9, 41-53.

Wasserman, A. (1981) In Chikofsky, E. (1989) How to lose productivity with productivity tools. *Proceedings of 3rd IFAC/IFIP Workshop*, Indiana, US, 1-4.

Wing, J. and Zaremski, A. (1991) Unintrusive ways to integrate formal specifications in practice. In *VDM '91: Formal Software Development Methods*, Prehn, S. and Toetenel, W. (Eds), Vol. 1, Springer-Verlag.

Wood-Harper, A. and Fitzgerald, G. (1982) A taxonomy of current approaches to systems analysis. *The Computer Journal*, **25**, 1, 12-16.

Yeh, R. (1991) System development as a wicked problem. *International Journal of Software Engineering and Knowledge Engineering*, **1**, 2, 117-130.

Yourdon, E. (1991) Sayonara, once again, structured stuff. *American Programmer*, **4**, 11, 40-49.

Yourdon, E. and Constantine, L. (1977) *Structured Design*, Yourdon Press, New York.

Zave, P. (1984) The operational versus the conventional approach to software development. *Communications of the ACM*, **February**, 104-118.

Zelkowitz, M. (1988) Resource utilisation during software development. Journal of Systems and Software, 8, 331-336.

Table 1: Summary of Research on System Development Methodologies

CRIS reviews:

CRIS--Comparative Review of Information Systems design methodologies represents a task group set up within IFIP Working Group 8.1. This task group was established in the early 1980s and its objective was to review available methodologies, conduct a feature analysis of available methodologies, and finally, to provide a synthesis of available methodologies, thus clarifying the issue of what methodologies are appropriate in different situations

(Olle *et al*., 1982; 1983; 1986; 1991)

Conceptual studies:

Methodology taxonomies
Methodology comparisons
Feature analyses
Frameworks for evaluating methodologies

(Avison & Fitzgerald, 1988; Banbury, 1987; Bantleman & Jones, 1984; Colter, 1984; Davis, 1982; Davis *et al*., 1988; Giddings, 1984; Gremillion & Pyburn, 1983; Hackathorn & Karimi, 1988; Maddison *et al*., 1984; McDonald *et al*., 1986; Peters & Tripp, 1977; Shemer, 1987; Song & Osterweil, 1992; Wood-Harper & Fitzgerald, 1982; Yeh, 1991)

Empirical studies of development approaches:

(Curtis *et al*., 1988; Gould & Lewis, 1985; Guimares, 1985; Hirschheim, 1985; Jenkins *et al*., 1984; Mahmood, 1987; Necco *et al*., 1987; Soloway *et al*., 1983; Sumner & Sitek, 1986; Vitalari & Dickson, 1983)

Table 2: Summary of Issues Supporting Formalised System Development Methodologies

Conceptual basis: Based on scientific paradigm Development process more amenable to project management and control, thus minimising risk and uncertainty Economic rationale: skill specialisation and elimination of irrational activities Epistemological rationale: provide a structural framework for the acquisition of knowledge
Pressures for increased formalism: Desirability of ISO-certification Government SDM standards: SSADM (UK, Ireland, Malta, Hong Kong, Israel) Dafne (Italy) Merise (France) NIAM (Netherlands) Department of Defense Std. 2167 (US) Software Capability Evaluation programme Growing interest in formal mathematical methods for systems development e.g. VDM, Gist, PAISLey, Z

Table 3: Summary of Issues Against Formalised System Development Methodologies

Definitional problems: Problems as to what exactly constitutes a SDM Differing philosophies, objectives and areas of focus for different SDMs. Major differences between some methodologies and trivial differences between others Generalisation from limited practical experience
Inadequacies of scientific paradigm: Systems development is not actually a rational process but most methodologies view it as rational Over-emphasis on technical aspects at the expense of softer social aspects
Shortcomings of the waterfall life-cycle: Often applied in a dogmatic and ironclad manner Linear sequential progression not an adequate reflection of the reality of systems development Requirements cannot typically be fully specified in advance
Goal displacement: Slavish adherence to SDM at the expense of actual systems development
Assumption that SDMs are universally applicable: Failure to recognise contingency factors and the uniqueness of every development situation
Inadequate recognition of people factors: SDMs do not cater for factors critical to successful development, such as individual creativity and intuition, or learning over time