18

A Classification of Methodological Frameworks for Computerized Information Systems Support in Organizations

John Krogstie
Andersen Consulting phone: +4722928200, fax: +4722928900, email: John.Krogstie@ac.com

Arne Sølvberg
Faculty of Electrical Engineering and Computer Science
The Norwegian Institute of Technology
University of Trondheim, Norway

Abstract

Although many conceptual frameworks for development and maintenance of information systems in organizations have been proposed, we experience a lack of integrated support of the evolutionary nature, the interconnectedness, and the social processes for developing such systems. This paper present a classification of methodological frameworks for evaluating important aspects of methodologies having this in mind. Contrary to most classification frameworks presented in literature which look solely upon different ways of supporting development of new information systems, we have in our framework a broader view, including larger parts of what we term computerized information systems (CIS) support in organizations.

In the end of the paper, we present the result of classifying a set of approaches to CIS-support in organizations described in academia and practice. No methodology is found to be sufficient in all respects, although newer approaches take more aspects into account.

Keywords

Methodology, classification

1 INTRODUCTION

Several frameworks for the classification of methodological frameworks have been developed through the years e.g. (Blum, 1994; Davis, 1988; Lyytinen, 1987). A weakness of these is in our view their limited scope, basically looking upon the development of a single application system in a comparatively stable environment. Organizations are continuously under the pressure of change from both internal and external forces. Most organizations of some size are supported by and depend upon a portfolio of application systems who likewise has to be changed, often

S. Brinkkemper et al. (eds.), *Method Engineering*

in a comparatively stable environment. Organizations are continuously under the pressure of change from both internal and external forces. Most organizations of some size are supported by and depend upon a portfolio of application systems who likewise has to be changed, often rapidly, for the organization to be able to keep up and extend their activities. The portfolio usually consist of a set of individual, but often highly integrated application systems whose long term evolution should be looked upon as a whole. Change is the norm, not the exception for both portfolios and their individual information systems (Alagappan and Kozaczynski, 1991; Williams et al., 1988). A first step towards facing this is to accept change as a way of life, rather than as an untowarded and annoying exception.

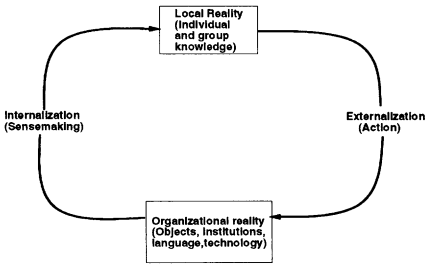

Figure 1: Social construction in an organization.

According to (Gjersvik, 1993) organizations are realities socially constructed through the joint actions of all the social actors in the organization. This process is illustrated in Figure 1. An organization consists of individuals who see the world in a way specific to them. The *local reality* is the way the individual perceives the world that he or she acts in. When the social actors of an organization act, they *externalize* their local reality. The most important ways social actors externalize their local reality, are to speak and to construct languages, artifacts, and institution. What they do is to construct *organizational reality*: To make something that other actors have to relate to in their work. This organizational reality may consist of different things, such as institutions, language, knowledge, artifacts and technology. Finally, *internalization* is the process of making sense out the organizational reality and making this part of the individual local reality.

We claim that the evolutionary aspects of computerized information systems (CIS) support are insufficiently covered by traditional approaches and tools. In addition, the process of social construction of the organizational reality is neglected in most development methodologies.

On this background, we will in this paper present a classification of methodological frameworks that takes also these aspects into account. We will in the end of the paper present the main results from an evaluation of a host of methodologies using the framework.

2 CLASSIFICATION OF METHODOLOGIES FOR COMPUTERIZED INFORMATION SYSTEMS SUPPORT

When deciding on relevant dimensions for a classification frameworks, we have tried to ask the questions *why, when, what, how, who, where* and *for how long* in the context of CIS support and

- *Why* do we attack the problem as we do? This is covered by the "Weltanschauung", i.e. underlying philosophical view of the methodology.
- *When* is the methodology applied? We have termed this aspect coverage in process indicating the main tasks that are covered by the methodology.
- *What* part of the portfolio is supported by the methodology? We have termed this aspect coverage in product.
- *How* do it help achieving the goals of CIS support? Based on the discussion in the introduction, we have concentrated on reuse and representation of product and process in the methodology with emphasis on conceptual modeling.
- *Who* is involved and where do changes take place? This is discussed under the area of stakeholder participation.
- *For how long* has the methodology been used. We term this aspect maturity: Is the methodology mature, being used for a long time in many organizations, with tool-support and support for evolution of the methodology.

Below, we will define and discuss each area in more detail.

2.1 "Weltanschauung":

FRISCO (FRISCO, 1995) differentiate between three different views:

- Objectivistic: "Reality" exists independently of any observer and merely needs to be mapped to an adequate description. For the objectivist, the relationship between reality and models thereof is trivial or obvious.
- Constructivistic: "Reality" exists independently of any observer, but what each person possess is a restricted mental model only. For the constructivist, the relationship between "reality" and models of this reality are subject to negotiations among the community of observers and may be adapted from time to time.
- Mentalistic: To talk about "reality" as such does not make sense because we can only form mental constructions of our perceptions. For the mentalist, what people usually call "reality" as well as its relationship to any model of it is totally dependent on the observer.

The methodologies that is found in literature can be characterized as being objectivistic or constructivistic. The "Weltanschauung" of a methodology is often not explicitly stated, but often appears only indirectly. Since different underlying philosophies may lead to radically different approaches, it is important to establish this. The distinction into objectivistic and constructivistic is parallel to the distinction between objectivistic and subjectivistic in the overview of Hirschheim and Klein (Hirschheim and Klein, 1989). Hirschheim and Klein also distinguish along the order-conflict dimension. In this dimension, the order or integration view emphasizes a social world characterized by order, stability, integration, consensus, and functional coordination. The conflict or coercion view stresses change, conflict, disintegration, and coercion. These two dimensions were originally identified by Burrel and Morgan (Burrel and Morgan, 1979) in the context of organizational and social research.

Based on the discussion in the introduction, it should come as no surprise that we find it beneficial to adapt a constructivistic world-view. Both the order and the conflict view combined with constructivism acknowledges a situation of continuous change.

2.2 Coverage in process

Do the methodology address:

- Planning of CIS-support
- Development of application systems
- Use and operation of application systems
- Maintenance of application systems

One or more of the above areas can be covered, more or less completely and in varying degrees of detail. More detailed specifications of dimensions of development methodologies are given by Blum (Blum, 1994), Davis (Davis, 1988) and Lyytinen (Lyytinen, 1987). Whereas Davis classifies a methodology according to the way it is able to address varying user-needs over time, Blum classifies development methodologies in two dimensions; if they are product or problem-oriented, and if they are conceptual or formal. We will only look upon the use of conceptual models and if these models are formal or not below. The product vs problem-oriented dimension as discussed by Blum is in our view a distinction on the *part* of development that is covered. Generally, every more detailed effort can be looked upon as a modeling task, where we differentiate based on the domain of modeling (Krogstie, 1995).

- The existing IS as it is perceived.
- The future IS as it is perceived.
- The future CIS as it is perceived.
- The (future) CIS in itself.

This correspond to what (Davis, 1995) term *understand problem*, *specify external behavior*, *design system*, and *implement system* respectively.

Lyytinen includes aspect covered by "Weltanschauung" and representation of product and process, in addition to linking technical, linguistic, and organizational aspects in a development methodology.

We claim that a comprehensive methodology should cover both planning, development, use, and maintenance in an integrated manner. The emphasis will be put on development and maintenance, but also the usage aspect is important, enabling the different end-users to make sense of the existing applications system in the organization, to both be able to use them more efficiently, and to be able to come up with constructive change-request and ideas for more revolutionary changes in the CIS support of the organization when the environment of the organization is changing. Planning aspects are important to be able to link the CIS-support of the organization up to strategic planning efforts in the organization, both to be able to implement the strategic plan, and to exploit information technology to the fullest in continuous development of the organization.

We claim that it is beneficial to not differentiate between development and maintenance in most cases, having a released based approach to CIS-support. This is partly based on figures appearing in our survey-investigation and in accompanying work presented in (Krogstie, 1995).

Maintenance has traditionally been looked upon as a more boring and less challenging task than development (Glass, 1992). Even if there are indications that this view might be changing e.g. (Layzell and Macauley, 1994) this still appears to be the prominent view among practitioners. According to our discussion in the introduction, it is both natural and desirable for CISs to change. As shown both in our own and other surveys, approximately half of the work which is normally termed maintenance is in fact further development of the information systems portfo-

lio, and should be given credit as such. On the other side, almost half of the new systems being developed are replacement systems, not extending what the users can do with the portfolio of systems. Thus seen from the end-users point of view, a better assessment of information system support efficiency seems to be found by blurring the old temporal distinction between maintenance and development. This is difficult to achieve when having a large mental and organizational gap between development and maintenance, even though the actual tasks being done have many similarities.

Swanson in (Swanson and Beath, 1989) recognizes the similarities of the tasks of development and maintenance, but still argues for keeping the old distinction based on the following perceived differences:

- As also noted in (Glass, 1992), a large proportion of traditional maintenance work is to perform un-design of existing systems, finding out what the system does. We will argue that with modern development approaches where as much as possible of the work should take place on a specification and design level, the difference will be smaller. Supporting this is the results of a survey reported on in (Dekleva, 1992b) which gave no conclusive evidence that organizations using modern development methods used less time on maintenance activities. On the other hand, time spent on emergency error corrections as well as the number of system failures decrease significantly with the use of modern development methods. Systems developed with modern methodologies seemed to facilitate making greater changes in functionality as the systems aged, and the request from users seemed more reasonable, based on a more complete understanding of the system. We also note that because of the large amount of replacement work of often poorly documented application systems, code understanding problems are often just as important when developing "new" systems as when maintaining old systems today. Code and design understanding will also often be an issue when reusing the products from other projects, and during traditional development, when due to changing work load, developers have to work on other peoples code for instance during system-test.
- It is generally believed that "Maintenance of systems is characterized by problems of unpredictable urgency and significant consequent fire-fighting. In difference to new systems development, which is buffered from the day to day tasks of the users, the systems in production is much more visible" (Swanson and Beath, 1989). First of all, also development projects with tight schedules has its share of fire-fighting. Traditionally, it has been found that approximately 20% of the maintenance work is corrective maintenance (Lientz and Swanson, 1980), and our result of 26% seems to build up on the importance of this. On the other hand, if we look upon the percentage of work that is performed to do immediately necessary corrective maintenance on the application level, we found in our own investigation (Krogstie and Sølvberg, 1994) a percentage of 6%, the similar figure in Lientz/Swanson being 12%. The total amount on corrective maintenance on the individual systems in our investigations was 15%. (Jørgensen, 1994) indicate that the assessed corrective percentage of the work used on maintenance often might be exaggerated since these kind of problems are more visible for management. They found in their investigation of individual maintenance tasks that even if 38% of the changes were corrective, this took only up 9% of the time used for maintenance. Management assessed the percentage of corrective maintenance to be 19%. Those managers who based their answers on good data had a result of 9% corrective maintenance. Also in our investigation, we found a similar tendency, on the

data of the maintenance task of the individual systems, those reporting to have good data, reported that only 8% of the work effort was corrective maintenance, 4% being emergency fixes. The same effect on over-assessing the amount of corrective maintenance has been reported earlier in (Arnold and Parker, 1982).

The problem of many small maintenance tasks done more or less continuously seems to be increased by how maintenance is often done, in an event-driven manner. In the Jørgensen investigation (Jørgensen and Maus, 1993), where 38% of the tasks were of an corrective nature, as much as 2/3 of the tasks where classified to have high importance by the maintainers themselves. The problem of changing priorities as described by Dekleva (Dekleva, 1992a) is closely related to this.

Even if the problem of emergency fixes seems to be smaller than earlier perceived, a methodology uniting development and maintenance must take into account that one has to be able to perform rapid changes to software artifacts.

2.3 Coverage in product

Is the method concerned with the development, use, and maintenance of

- One single application system.
- A family of application systems.
- The whole portfolio of application systems in an organization.

Also finer classifications can be perceived, i.e. methodologies that are specifically geared towards the use of specific technology, or to solve problems within specific domains, but we regard this as extensions of a general methodology rather than as independent methodologies. We will argue that it is beneficial for a complete methodology to be able to consider the whole portfolio in an integrated manner and not only the single application system. For the end-users, it is not important which application system that is changed. What is important is that their perceived needs are supported by the complete portfolio. This do obviously not mean that one always need to consider the whole portfolio when enhancing the CIS-support of the organization.

Application systems are not developed in a vacuum. They are related to old systems, by inheriting data and functionality, and they are integrated to other systems by data, control, presentation philosophy, and process (Thomas and Nejmeh, 1992). As reported in our investigation, the most important reason for replacements apart from systems being unmaintainable, was integration of application systems. Often when doing this kind of integration, it can be useful to collect the functionality of several existing application systems into a new application system, something which is not well supported when having strict borders for what is regarded as inside and outside of an application system.

As noted in (Swanson and Beath, 1989) the CISs of an organization tend to congregate and develop as families. By original design or not, they come to rely upon each other for their data. In Swanson/Beath 56% of the systems where connected to other systems through data integration. In our survey, we found that 73% of the main information systems in the organizations surveyed were dependent on data produced by other systems. In 40% of the responses to this question *all* the main system which the organization depended upon on a daily basis were dependant on data produced by other systems.

Over time, newer application systems originate in niches provided by older ones, and identifiable families of systems come to exist. Relationships among families are further established.

In the long run, an organization is served more by its CISs as a whole than it is by the application systems taken individually.

2.4 Reuse of product and process

Reusing experience is a key ingredient to progress in any discipline. Without reuse everything must be re-learned and recreated; progress in an economical fashion is unlikely. The need to utilize extensive reuse is based on the need for evolutionary and rapid changes in the CIS of an organization as discussed in the introduction.

An comprehensive overview of dimensions of reuse is given y (Prieto-Diaz, 1993):

- By substance: The essence of what is reused:
  - Idea reuse involves reusing formal notions, such as a general solution to a class of problems.
  - Artifacts reuse: Examples of artifacts are code, conceptual models, design, specifications, objects, text, architectures, and test data.
  - Procedures reuse: Formalizing and encapsulating software development procedure. Procedures reuse also means reusing skills and know-how, i.e. having a development and maintenance methodology can be looked upon as reuse in this sense.
- By scope: The form and extent of reuse:
  - Vertical reuse is reuse within the same application area.
  - Horizontal reuse is reuse across application areas.
- By mode: How reuse is conducted:
  - Planned reuse: The systematic and formal practice of reuse. Guidelines and procedures for reuse have been defined, and metrics are being collected to assess reuse performance.
  - Ad-hoc reuse: An informal practice, in which components are selected from general libraries.
- By technique: How reuse is implemented:
  - Compositional reuse is the use of existing artifacts as building blocks for new systems.
  - Generative reuse is reuse at the specification level by means of design and code-generators.
- By intention: Defines how elements will be reused:
  - As-is or black-box reuse is reuse without modifications.
  - Modified or white-box reuse involves modifications of what is reused.

It is usual to differentiate between methodologies being *for reuse* and those being *with reuse* (Karlsson (ed.), 1995; Wilkie, 1993). Another distinction is between reuse-in-the-large and reuse-in the-small, where reuse in the large refers to the use of packaged solutions and frameworks. We will restrict the use of the term in the evaluations to include the planned reuse of artifacts, i.e. not including that using a methodology is an example of reusing procedures.

2.5 Representation of product and process

Knowledge about the process and the product of CIS development and maintenance can be represented using different kinds of languages. These languages can be informal, semi-formal, or

formal, having a logical and/or a executional semantics. These terms are defined as follows:

Language : A set of *symbols*, the **graphemes** of the language being the smallest units in the writing system capable of causing a contrast in meaning, a set of **words** being a set of related *symbols* constituting the **vocabulary** of the language, rules to form **sentences** being a set of related words (**syntax**), and some inter-subjectively agreed definitions of what the different sentences mean (**semantics**).

A **formalism** is a formal *language*, i.e. a *language* with a precisely defined *vocabulary*, *syntax*, and *semantics*. If the semantics is based on mathematical logic, we use the term **logical formalism**. If it is possible to execute a set of sentences in the language on a computer, the language is said to have an operational semantics.

A **semi-formal language** is a *language* with a precisely defined *vocabulary* and *syntax*, but without a precisely defined *semantics*.

We will in this paper concentrate on conceptual modeling languages. As will be illustrated, conceptual modeling is believed to be an important technique for CIS support in organizations when combining development and maintenance having support for not only a single application systems, but the whole application system portfolio, being based around social construction theory and reuse. When discussing the benefits of using conceptual modeling below, we should have in mind that we are primarily talking about partly graphical languages which are semi-formal or formal, have a limited vocabulary, and which can be used in many areas on varying levels of formality and completeness.

- A conceptual model has the possibility of being a problem-oriented description of the requirements for CIS support, without being restrained too early by technical constraints. In this way we believe one can more easily support a process of social construction of information systems. A problem-oriented approach has been asked for by many researchers (Borgida et al., 1985; Bubenko jr., 1983; Hagelstein, 1988; van Assche et al., 1988) and conceptual modeling is looked upon as one way of achieving this.
- Due to the visual nature of many conceptual modeling languages they are believed to be more helpful in the sense-making process of what is modeled than the model which is implicit in the code of an application system. On the other hand, if we want to refine the conceptual models into a form that is suitable for automatic code-generation, the essential difficulty of complexity discussed by Brooks (Brooks Jr., 1986) will again appear.
- Since the separate conceptual modeling languages only include a limited set of phenomena, this enable a focusing of concerns, and it is possible to deduce properties that are difficult if not impossible to perceive directly, by concentrating on only some aspect at the time. This is obviously also problematic if this makes one blind for other concern, or makes it impossible to externalize certain explicit knowledge. Based on this we will claim that one need a set of interrelated semi-formal and formal modeling languages which can cover different perspectives for conceptual modeling to be more generally useful.
- Conceptual models developed in early parts of development can be used as an outset for further design and implementation, supporting generative reuse. Conceptual models are also believed to be easier to maintain than textual documents that do not have any other mission than to serve as documentation, since they can be constructed as part of the process of developing and maintaining the application system in the first place, thus supporting

change and an integration of development and maintenance techniques. It is also easier to get an overview of the CIS-support of an organization if the languages for conceptual modeling are known and sufficient tool support for handling them exist, thus potentially supporting the long range planning and evolution of the whole portfolio.

According to our survey (Krogstie, 1995), most of the organizations having started to use CASE-tools for development and maintenance, use these for conceptual modeling.

2.6 Stakeholder participation

In general, stakeholders in CIS-support can be divided into the following groups (Macauley, 1993):

- Those who are responsible for its design, development, introduction and maintenance, for example, the project manager, system developers, communications experts, technical authors, training and user support staff, and their managers.
- Those with financial interest, responsible for the application systems sale or purchase.
- Those who have an interest in its use, for example direct or indirect users and users managers.

We focus here specifically on end-user participation.

A **user** of a *CIS* is defined as a person who potentially increases his *knowledge* about some *phenomena* other than the *CIS* with the help of the *CIS*. An **end-user** increases his and hers *knowledge* in areas which are *relevant* to him by *interacting* with the *CIS*. **Indirect users** increase their *knowledge* by getting results from the *CIS* without *interacting* directly with the *CIS*.

This is somewhat different from how 'user' is often defined, terming the system development and maintenance personell as 'primary users' (Hirschheim, 1984) or technical users. Not including these persons as users in the following discussion do not mean that they are not important stakeholders.

The term 'participation' means to take part in something. There exists different forms of participation:

- **Direct participation:**
  Every stakeholder has an opportunity to participate.
- **Indirect participation:**
  Every stakeholder participate more or less through representatives that are supposed to look after their interests. The representatives can either be:
  - Selected: The representatives are picked out by somebody, e.g. management.
  - Elected: The representatives are chosen from among their co-workers.

Many arguments for having participation have been given in the literature see e.g. (Greenberg, 1975; Mumford, 1983) for classifications. Here, user participation is basically motivated through a cost-benefit-perspective on the long run. Since all stakeholders have their individual local reality, everyone have a potential useful view of how the current situation can be improved. Including more people in the process will ideally increase the possibility of keeping up with the ever more rapidly changing environment of the organization. Added to this is the general argument of including those who is believed to have relevant knowledge in the area, and which are influenced by the solution. As indicated in several surveys, general participation appears to be a general indicator for (development) project success as perceived by all the different stakeholders. In Bergersen (Bergersen, 1990), the three most important factors for overall perceived

project success were found to be the goal-setting, management support, and user-participation. In van Swede (van Swede and van Vliet, 1994) the main contributions of success in the sense of satisfaction of all stakeholders were a cooperative environment, presence of a win-win starting point by considering the interest of all stakeholder-group, quality of project staff, and quality of project management.

According to Heller (Heller, 1991), participation is sharing power and influence. He has divided the degree of influence and power into six categories as illustrated in Figure 2.

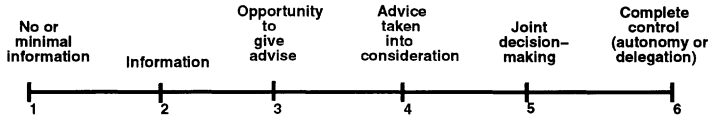

Figure 2: Scale of influence and power

We would claim that participation when applied should be in categories 4, 5, or 6 on this scale, and we will use this scale when classifying methodologies according to this aspect.

Due to the large number of potential stakeholders in a development effort, in most cases representative participation will be the only practical possibility. From the point of view of social construction, it is doubtful that a user representative can truly represent anyone else than himself. On the other hand, even if the internal reality of each individual will always differ to a certain degree, the explicit knowledge concerning a constrained area might be more or less equal, especially within groups of social actors (Gjersvik, 1993; Orlikowski and Gash, 1994). Another factor is the scope of participation, i.e. when do participation take place. Usually one would expect that user-participation would take place heavily in analysis and in acceptance testing, more lightly in design, and very little in implementation, but this will often depend on the chosen methodology. When it comes to suggesting improvements of the current information system of the organization, direct participation should be possible. Also in planning leading up to project establishment, a larger proportion of the stakeholders should be able to participate.

Another aspect related to this point, is where the changes of the portfolio takes place:

- In the user organization.
- In a data department, developing customized systems.
- Centrally, with one unit developing the core of the systems, which are then customized locally.
- Externally developed packages with large local adaptions.
- Externally developed packages with small local adaptions.
- By a different organization all-together (outsourcing)

Typically, one would expect a mix of these models within the support of a portfolio. We will not investigate this in detail here.

2.7 Maturity

Whereas some of the methodologies being presented in literature have been used for many years by many organizations, others are only described in theory, and never tried out in practice. When

discussing the maturity of a methodology, we can differentiate between the following factors:

- Is the methodology properly described? (vs. representation of process)
- Is the methodology supported by mature, high quality tools?
- Is the methodology (re)used and updated through practical application? Is it used by many organizations, supporting a large part of the portfolios in these organizations?
- Is the methodology undergoing a conscious evolution based on experience with the use of the methodology, being "annotated" with information about what parts of the methodology seems appropriate in a given situation?

Different parts of a methodology will typically be of varying maturity.

3 SUMMARY AND CONCLUSION

We have in (Krogstie, 1995) given an overview and classification of a host of existing methodologies and frameworks using the above classification. The following methodologies and frameworks were classified primarily based on the cited works.

- The conventional waterfall model (Royce, 1970).
- The structured life cycle (Yourdon, 1988).
- Iterative and throwaway prototyping (Carey, 1990).
- Incremental development (Davis et al., 1988).
- Transformational and operational development (Zave, 1982).
- Tempora (Loucopoulos et al., 1991).
- Method/1 (METHOD1:89, 1989).
- The spiral model (Boehm, 1988).
- The hierarchical spiral model (Iivari, 1990a).
- The fountain model (Henderson-Sellers and Edwards, 1990).
- OMT (Rumbaugh et al., 1991).
- REBOOT (Karlsson (ed.), 1995).
- CONFORM (Capretz and Munro, 1994).
- Maintenance as reuse-oriented development (Basili, 1990).
- Multiview (Avison and Wood-Harper, 1990).
- STEPS (Floyd et al., 1989).
- Systems devtenace (Krogstie, 1995).

Due to space limitation, we will here only summarize this work, noting that we have tried to include examples such that all aspects are covered in full by at least one approach. A short summary of our classifications is given in Table 3. It includes the first six aspects of the classification. The first column indicate the "Weltanschauung" as judged by what we have read on the methodologies. Process coverage indicates the areas that we judge the methodologies give comprehensive support in. Product coverage differentiate upon those methodologies which we regard as being useful for the support of more than one CIS at the time. Reuse is not discussed explicitly in many methodologies, which is indicated with - in the table. The 'conceptual modeling' column indicate the use and kind of languages used. 'OOA' refers to the use of languages for object oriented analysis, which are mostly semi-formal. Finally, the participation column indicates the strength of participation as indicated in the descriptions of the methodology.

Table 1 Classifications of methodologies

Methodology	*Weltanschauung*	*Process coverage*	*Product coverage*	*Reuse*	*Conceptual modeling*	*Part. (range)*
Waterfall	Objectivistic	Development +	one	-	Little	2-4
Structured	Objectivistic	Development	one	-	Semi-formal	2-4
Prototyping	Objectivistic	Development (early)	one	with	-	4-5
Operational	Objectivistic	Development	one	Generative	Formal	2-4
Tempora	Objectivistic	Development	one	Generative	Formal	2-4
Method/1	Objectivistic	Planning/ Development	one/ portfolio	In the large	Semi-formal	2-5
Spiral	Objectivistic	Development/ maintenance	one	with	-	3-5
Hierarchical spiral	Objectivistic	Development/ maintenance	one	-	Yes	3-4
Fountain	Objectivistic	Development (mainly)	one	for/with	OOA	-
OMT	Objectivistic	Development	one	for	OOA	2-4
REBOOT	Objectivistic	Development/ maintenance	one/ portfolio	for/with	OOA	2-3
CONFORM	Objectivistic	Maintenance	one	-	-	-
Basili	Objectivistic	Development/ maintenance	one/ portfolio	for/with	-	-
Multiview	Constructivistic	Development	one	-	Semi-formal	4-5
STEPS	Constructivistic	Development maintenance/use	one	-	-	4-5
Devtenance	Constructivistic	Development maintenance/use	One/ portfolio	Generative	Formal	4-5

According to our classifications we conclude the following:

- Weltansscahuung: As also noted in (Hirschheim and Klein, 1989), most earlier and current methodologies for application systems development and maintenance have an objectivistic outlook. Some exceptions illustrated are STEPS (Floyd et al., 1989), Multiview (Avison and Wood-Harper, 1990), and systems devtenance (Krogstie, 1995). Other examples are methodologies based on SSM (Checkland, 1981) and some PD-methodologies (Schuler and Namioka, 1993).
- Coverage in process: Most methodologies for CIS-support are focused on development, with maintenance being looked upon as a separate end-phase if considered at all. Several methodologies focused on maintenance also exist (e.g.CONFORM (Capretz and Munro, 1994), see also (Boldyref et al., 1994)), even if this part of CIS-support is not shown the same interest as development by researchers according to (Hale et al., 1990; Jørgensen, 1994). Some methodologies covers both development and maintenance in the same framework in an integrated manner (e.g. The Spiral Model (Boehm, 1988), the Hierarchical Spiral Model (Iivari, 1990a; Iivari, 1990b) and the framework presented by Basili (Basili, 1990) where also emergency error-correction is covered). STEPS (Floyd et al., 1989) and systems devtenance (Krogstie, 1995) also includes the usage aspect. Method/1 includes IT-planning in an integrated manner.
- Coverage in product: We have found few methodologies apart from system devtenance that cover traditional development or maintenance of the whole portfolio in a focused manner, even though maintenance can be said to often be performed in this way (Swanson and Beath, 1989). Several methodologies include organization-wide CIS-planning (e.g. METHOD/1 (METHOD1:95, 1995)).
- Reuse: Some methodologies explicitly addressing reuse exist (e.g. REBOOT (Karlsson (ed.), 1995)), even if few development and maintenance methodologies are geared towards conscious component reuse. Operational and transformational approaches as described in (Zave, 1982) are highly geared towards generative reuse. This is also the case with Tempora and systems devtenance.
- Use of conceptual models: Many methods use conceptual modeling to some extent, even if most use only semi-formal modeling languages. On the other hand, the use of operational conceptual models have received increasing interest as illustrated through Tempora and systems devtenance.
- Stakeholder participation: Increasingly looked upon as important both in objectivistic and especially constructivistic methodologies. This might be endangered by the current trend of more and more use of packages and outsourcing, although this might have economic advantages in the short run.
- Maturity: Most mature methodologies resembles traditional waterfall, but many of these are taking newer aspects into account e.g. Method/1 and extensions of this. Most methodological frameworks described in literature have a very low maturity. This especially applies to system devtenance, which is the framework which otherwise are meant to best cover the other six aspects.

4 CONCLUDING REMARKS

There seems to be an overall view that there are no right detailed methodology for all situation (Avison and Wood-Harper, 1990; Floyd et al., 1989; Glasson, 1989; Iivari, 1990a) something which are also recognized in more traditional methodologies like Method/1. The different development and maintenance efforts can vary according to several factors e.g.:

- The complexity of the application system (cf. (Brooks Jr., 1986)).
- The current rate of change(cf. the discussion on evolution in the introduction).
- The size, perceived importance, and risks of performing the changes (cf. (Boehm, 1988)).
- The number of stakeholders affected, skills needed and possessed.
- The number of different views of the situation (cf. social construction theory as described in the introduction).

Thus there is a need for flexibility, but in our opinion one still need a methodological framework of some sort to be able to deliver CIS-support in an organization. Taking into account the multitude of techniques, there is an obvious need for an integrative framework that can incorporate existing more detailed approaches and support their flexible situation-dependant use. The work presented in this paper is meant to give an indication of some of the main aspects that such a framework should cover.

Taking a philosophical standpoint neither reuse nor conceptual modeling nor having a defined methodology can be optimal, since all situations are unique, and thus in principle can best be attacked by using unique means. Reusing artifacts originally produced for some other purpose, in effect means to apply an externalization of the local reality of someone else than the current stakeholders, which thus can not be optimal. On the other hand, reuse is performed all the time. Using a commercial DBMS is for instance reuse, but it is not very wise to produce your own database management system when you perceive a need for this kind of functionality if you do not have very special needs. A balance between the different concerns brought up by our philosophical outlook is thus necessary.

5 REFERENCES

Alagappan, V. and Kozaczynski, W. (1991). The evolution of very large systems. In Lowry, M. R. and McCartney, R. D., editors, *Automating Software Design*, pages 1–24, California, USA. The MIT Press.

Arnold, R. S. and Parker, D. A. (1982). The dimensions of healthy maintenance. In *Proceedings of the 6th International Conference on Software Engineering (ICSE)*, pages 10–17. IEEE Computer Society Press.

Avison, D. E. and Wood-Harper, A. T. (1990). *Multiview: An Exploration in Information Systems Development.* Blackwell, Oxford, England.

Basili, V. R. (1990). Viewing maintenance as reuse-oriented software development. *IEEE Software*, 7(1):19–25.

Bergersen, L. (1990). *Prosjektadministrasjon i systemutvikling. Aktiviteter i planlegningsfasen som påvirker suksess (In Norwegian).* PhD thesis, ORAL, NTH, Trondheim, Norway.

Blum, B. I. (1994). A taxonomy of software development methods. *Communications of the ACM*, 37(11):82–94.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72.

Boldyref, C., Burd, E. L., and Hather, R. M. (1994). An evaluation of the state of the art for application management. In (Müller and Georges, 1994), pages 161–169.

Borgida, A., Greenspan, S., and Mylopoulos, J. (1985). Knowledge representation as the basis for requirements specification. *IEEE Computer*, 18(4):82–91.

Brooks Jr., F. P. (1986). No silver bullet. Essence and accidents of software engineering. In Kugler, H. J., editor, *Information Processing '86*, pages 1069–1076. North-Holland.

Bubenko jr., J. A. (1983). On concepts and strategies for requirements and information analysis. In *Information Modelling*, pages 125–169. Chartwell-Bratt Ltd.

Burrel, G. and Morgan, G. (1979). *Sociological Paradigms and Organizational Analysis*. Heinemann.

Capretz, M. A. M. and Munro, M. (1994). Software configuration management issues in the maintenance of existing system. *Journal of Software Maintenance*, 6:1–14.

Carey, J. M. (1990). Prototyping: Alternative systems development methodology. *Information and Software Technology*, 32(2):119–126.

Checkland, P. B. (1981). *Systems Thinking, Systems Practice*. John Wiley & Sons.

Davis, A. M. (1988). A comparison of techniques for the specification of external system behavior. *Communications of the ACM*, 31(9):1098–1115.

Davis, A. M. (1995). Object-oriented requirements to object-oriented design: An easy transition? *Journal of Systems and Software*, 30(1/2):151–159.

Davis, A. M., Bersoff, E. H., and Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, 14(8):1453–1461.

Dekleva, S. M. (1992a). Delphi study of software maintenance problems. In *Proceedings of the Conference on Sofware Maintenance (CSM'92)*, pages 10–17.

Dekleva, S. M. (1992b). The influence of the information systems development approach on maintenance. *MIS Quarterly*, pages 355–372.

Floyd, C., Reisin, F.-M., and Schmidt, G. (1989). STEPS to software development with users. In Ghezzi, C. and McDermid, J. A., editors, *2nd European Software Engineering Conference (ESEC'89)*, pages 48–63, University of Warwick, Coventry, England.

FRISCO (March 1995). Personal communication with the FRISCO task group.

Gjersvik, R. (1993). *The Construction of Information Systems in Organization: An Action Research Project on Technology, Organizational Closure, Reflection, and Change*. PhD thesis, ORAL, NTH, Trondheim, Norway.

Glass, R. L. (1992). We have lost our way. *Journal of Systems and Software*, 18(2):111–112.

Glasson, B. C. (1989). Model of system evolution. *Information and Software Technology*, 31(7):351–356.

Greenberg, E. S. (1975). The consequences of worker participation: A clarification of the theorethical litterature. *Social Science Quarterly*, 56(2).

Hagelstein, J. (1988). A declarative approach to information systems requirements. *Knowledge Based Systems*, 1(4):211–220.

Hale, D. P., Haworth, D. A., and Sharpe, S. (1990). Empirical software maintenance studies during the 1980s. In *Proceedings of the Conference on Software Maintenance (CSM'90)*, pages 118–123. IEEE Computer Society Press.

Heller, F. (1991). Participation and competence: A necessary relationship. In Russel, R. and Rus, V., editors, *International Handbook of Participation in Organizations*, pages 265–281.

Henderson-Sellers, B. and Edwards, J. M. (1990). The object-oriented systems life cycle. *Communications of the ACM*, 33(9):142–159.

Hirschheim, R. A. (1984). A participative approach to implementing office automation. In *Proceedings from the Joint International Symposium on Information Systems*, pages 306–329, Sydney, Australia.

Hirschheim, R. A. and Klein, H. K. (1989). Four paradigms of information systems development. *Communications of the ACM*, 32(10):pages 1199–1216.

Iivari, J. (1990a). Hierarchical spiral model for information system and software development. Part 1: Theoretical background. *Information and Software Technology*, 32(6):386–399.

Iivari, J. (1990b). Hierarchical spiral model for information system and software development. Part 2: Design process. *Information and Software Technology*, 32(7):450–458.

Jørgensen, M. (1994). *Empirical studies of Software Maintenance*. PhD thesis, Department of Informatics, University of Oslo, Oslo, Norway.

Jørgensen, M. and Maus, A. (1993). A case study of software maintenance tasks. In *Proceedings of Norsk Informatikk Konferanse 1993 (NIK'93)*, pages 101–112, Halden, Norway.

Karlsson (ed.), E.-A. (1995). *Software Reuse: A Holistic Approach*. John Wiley & Sons.

Krogstie, J. (1995). *Conceptual Modeling for Computerized Information Systems Support in Organizations*. PhD thesis, IDT, NTH, Trondheim, Norway.

Krogstie, J. and Sølvberg, A. (1994). Software maintenance in Norway: A survey investigation. In (Müller and Georges, 1994), pages 304–313. Received "Best Paper Award".

Layzell, P. J. and Macauley, L. (1994). An investigations into software maintenance - perception and practices. *Software Maintenace: Research and Practice*, 6:105–119.

Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management*. Addison Wesley.

Loucopoulos, P., McBrien, P., Schumacker, F., Theodoulidis, B., Kopanas, V., and Wangler, B. (1991). Integrating database technology, rule-based systems and temporal reasoning for effective information systems: The TEMPORA paradigm. *Journal of Information Systems*, 1:129–152.

Lyytinen, K. (1987). A taxonomic perspective of information systems development: Theoretical constructs and recommendations. In Boland Jr, R. J. and Hirschheim, R. A., editors, *Critical Issues in Information Systems Research*, chapter 1, pages 3–41. John Wiley & Sons.

Macauley, L. (1993). Requirements capture as a cooperative activity. In *Proceedings of the First Symposium on Requirements Engineering (RE'93)*, pages 174–181.

METHOD1:89 (1989). *FOUNDATION - Method/1, Tools Reference Manual, Version 2.1*. Andersen Consulting.

METHOD1:95 (1995). *Method/1, System Development Management*. Andersen Consulting.

Müller, H. A. and Georges, M., editors (1994). *Proceedings of the International Conference on Software Maintenance (ICSM'94)*. IEEE COmputer Society Press.

Mumford, E. (1983). Participation - from Aristotle to today. In Bemelmans, T. M. A., editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 95–104. North-Holland.

Orlikowski, J. W. and Gash, D. C. (1994). Technological frames: Making sense of information technology in organizations. *ACM Transactions on Information Systems*, 12(2):174–207.

Prieto-Diaz, R. (1993). Status report: Software reuseability. *IEEE Software*, pages 61–66.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. In *Proceedings WESCON*.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ.

Schuler, D. and Namioka, A. (1993). *Participatory design: Principles and Practices*. Lawrence Erlbaum.

Swanson, E. B. and Beath, C. M. (1989). *Maintaining Information Systems in Organizations*. Wiley Series in Information Systems. John Wiley & Sons.

Thomas, I. and Nejmeh, B. A. (1992). Definitions of tool integration for environments. *IEEE Software*, 9(2):29–35.

van Assche, F., Layzell, P., Loucopoulos, P., and Speltincx, G. (1988). Information systems development: A rule-based approach. *Knowledge Based Systems*, 1(4):227–234.

van Swede, V. and van Vliet, H. (1994). Consistent development: Results of a first empirical study of the relation between project scenario and success. In Wijers, G., Brinkkemper, S., and Wasserman, T., editors, *Proceedings of the 6th International Conference on Advanced Information Systems Engineering (CAiSE'94)*, pages 80–93, Utrecth, Netherlands. Springer Verlag.

Wilkie, G. (1993). *Object-Oriented Software Engineering - The Professional Developers's Guide*. Addison-Wesley.

Williams, G. B., Mui, C. K., Johnson, B. B., and Alagappan, V. (1988). Software design issues: A very large information systems perspective. Technical report, CStar, Arthur Andersen, Chicago.

Yourdon, E. (1988). *Managing the System Life Cycle*. Prentice-Hall.

Zave, P. (1982). An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, 8(3):250–269.

6 BIOGRAPHY

- **John Krogstie** is a Senior Consultant with Andersen Consulting ANS in Norway. Krogstie received a MSc and a PhD in computer science from the University of Trondheim.
- **Arne Sølvberg** is a professor of computer Science at the University of Trondheim, NTNU. Sølvberg received a MSc in applied physics and a PhD in computer science from the University of Trondheim.