2 Methodologies for Computerised Information Systems Support in Organisations

In this chapter, we present the generic tasks and model types found in information systems development and evolution, and main methodologies for mixing different phases of information system development. In particular, we describe in more detail the main approaches to model-based development presented on a high level in Chap. 1, but also provide a historical account of the development of methodologies in the area of information systems development and evolution. Methodologies are classified relative to goal, process, product, capabilities needed, stakeholder participation, organisation and location of the tasks to be done. The last section focuses on stakeholder involvement, in particular participatory development of models.

2.1 A Framework for IS methodologies

Several frameworks for classifications of methodologies have been developed through the years e.g. (Abrahamsson et al 2010, Berente and Lyytinen 2007, Blum 1994, Davis 1998, Lyytinen 1987). A weakness of these is in our view their limited scope, primarily looking upon the development of a single application system in a comparatively stable environment. IT has over the last 30 years gone from obscure to infrastructure. All organisations are dependent on an application systems portfolio supporting its current and future tasks, and newcomers in any area are dependent on establishing new such application portfolios quickly in a way that can evolve with changed business needs, technological affordances and expectations among co-operators, competitors and customers. Also all people interacting with the organisation expect to be able to do this through the means of IT applications.

Over these 30 years, one has regularly investigated how IT-systems are developed, used and evolved. In (Lientz et al. 1978), results from a 1977 survey on distribution of work on IT in organisations where published. Somewhat surprisingly at that time, one found that almost half of the time was used on maintenance (i.e. changing systems that were already in production). We have done similar investigations in Norway in 1993, 1998, 2003, and 2008. For some areas, there are large differences between the results of investigations. Here and below, we will in particular look the results from the 2008 investigation (Davidsen and Krogstie 2010) and compare some of the results with the results of similar investigations. The most important of these investigations are:

1. The investigation carried out by Krogstie, Jahr and Sjøberg in 2003 reported in (Krogstie et al. 2006).
2. The investigation carried out by Holgeid, Krogstie and Sjøberg in 1998 reported in (Holgeid et al. 2000).
3. The investigation carried out by Krogstie in 1993 reported in (Krogstie and Sølvberg 1994, Krogstie 1995, Krogstie 1996).
4. The Lientz and Swanson investigation (LS) (Lientz et al. 1978): That investigation was carried out in 1977, with responses from 487 American organisations on 487 application systems.
5. The Nosek and Palvia investigation (NP) (Nosek and Palvia 1990): A follow-up study to Lientz/Swanson performed in 1990 asking many of the same questions as those of Lientz and Swanson. Their results are based on responses from 52 American organisations.

As an example, Table 2.1 illustrates the developments in the main programming languages being used. The use of underlying technical platform (e.g. the database technologies used) has also changed a lot.

Table 2.1 Percentages of systems developed using different programming languages

	2008	2003	1998	1993	NS90	LS77
COBOL	4.5	0.5	32.6	49	51	52
4GL	12	13.5	16.9	24	8	
C	2.4	12.5	15.4	4	3	
C++	17.5	23.1	15.1			
C#	4.9					
RPG/ Scripts	6.7	NA	12.9	4	10	22
Java	22.6	29.8	2			
Ass.	0.4		0.9	3		12
Fortran			0.6	4	7	2.4
PASCAL			0.3	2		
PL/1			0.3	2		3.2
Other	28.9	20.2	2.6	6	21	7.7

Other areas are remarkably stable. As illustrated in Table 2.2, the split of the time use for development vs. maintenance is still as it was 30 years ago. (In absolute terms, less time is used for both development and maintenance, since most of the time used by IT-departments these days are on user-support and systems operations). Over the last 20 years, the percentage of so-called new systems that are in fact replacement systems, being installed basically to replace old systems, has stayed above 50%, rising slowly.

Table 2.2 Comparisons of maintenance figures in percent across investigations

Category	2008	2003	1998	1993	NP 90	LS 77
Development	21	22	17	30	35	43
Maintenance	35	37	41	40	58	49
Other work	44	41	42	30	7	8
Disregarding other work						
Development	34	34	27	41	38	47
Maintenance	66	66	73	59	62	53

On the other hand, most research on the use of IT to support organisations, information systems and software engineering has focused on the development of new systems, even if this kind of work is actually no longer typical for most IT-related activities. As different actors support organisations with IT in what can be looked upon as a flexible ecosystem of solution providers, also the actors being supported is more diverse. There are still a number of stable organisations (local municipalities comes to mind), but also these and (parts of) other organisations participate in an increasingly wider spectrum of organisational forms (e.g. virtual enterprises, value chain networks, extended enterprises, dynamic networked organisation etc), a trend that we look upon as likely to continue.

Organisations are continuously under the pressure of change from both internal and external forces. All organisations of some size are supported by and depend upon a portfolio of application systems that likewise has to be changed, often rapidly, for the organisation to be able to keep up and extend their activities. In addition to be integrated internally, more and more systems are integrated with systems outside the control of the organisation, and have to adhere to different standards and common components that are also potentially changing. Change is thus is the norm, not the exception for an organisation's application portfolios and their individual information systems. A first step towards facing this is to accept change as a way of life, rather than as an annoying exception.

With this backdrop, and our worldview of social construction of reality described in Chap. 1, we have classified methodologies according to the following areas:

- What is the *goal* of the methodology, and why do we attack the overall problem area in the way we do? This last aspect is covered by the "Weltanschauung", i.e. the underlying philosophical view of the methodology, whereas the first relates to the value one expect to achieve.
- When is the methodology applied? We have termed this aspect coverage in *process*, meaning the main tasks that are covered by the methodology.
- What part of the application portfolio of information systems is supported by the methodology? We have termed this aspect coverage in *product*.
- What are the *capabilities* needed to make the methodology work.
- *Who* is involved? This is discussed under the area of stakeholder participation.
- How is the work *organised* (i.e. what organisational actors are involved)
- Where are the changes done? This is covered under the aspect of *location.*

2.1.1 Goal of the Methodology

IT is normally seen as a means to achieve results in another sphere (e.g. achieve business results, better public services), rather than a goal in itself. What this external goal is will naturally vary, but it is normally to achieve some sort of *value*. Based on the goals of those using resources to perform the change we can briefly highlight these as:

- Ensure economic gain
- Ensure personal gain
- Ensure organisational (business) gain
- Ensure societal gain

Personal and societal gain might be linked to economic gain, but can raise a number of goals in addition which are not purely economical.

Economical value is highly tangible and can be viewed from different stakeholder perspectives. Business value is somewhat less tangible that includes all forms of value that determine the health and well-being of an organisation in the long-run. Business value expands the concept of economical value to include other forms of value such as e.g. employee value, customer value, supplier value, managerial value and potentially also societal value (related to corporate responsibility). Business value also often embraces intangible assets not necessarily attributable to any stakeholder group such as intellectual capital and a firm's business model and public opinion.

For an organisation, IT can be looked upon as a service. According to Kuusisto (2008), the concept of value adding services or products imply that value is contained in the product or the service. Another perspective is the value-in-use concept that focuses on the experience of a user interacting with products or services in use situations. This concept implies that the user is always a co-creator of the value. According to this concept, the user experience and perception are essential to be able to determinate value (Kuusisto 2008).

In all cases, one can differentiate three stages when using IT to achieve value:

1. Plan for value achievement. In a business setting this would often be termed 'develop business case and value realisation plan'.
2. Perform change (and manage the business case on the way, some of the basis for the original business case might change, and new opportunities might arise). Additional value can be explicit and easy to grasp, but some additional value is also tacit. Tacit value, e.g. the improved understanding of a work process for those involved in the installation of a work process support system, is often not explicitly captured in traditional project documentation, but may still affect decisions before or during a project, or the perceived value of the project in retrospect. Future reuse of results from the change can be a benefit of the current project, especially if this potential is taken into account at an early stage.
3. Implement change. First at this stage it is possible to achieve the expected value, and a focus on benefit realisation is necessary to assure that the value will be achieved over time.

The broadest set of issues to address is found relative to eGovernment solutions, and we will discuss briefly benefit realisation in this setting below, based on (Flak et al. 2009).

Calculating costs and benefits in the public sector is a major challenge. The reason for this is that there are difficulties in calculating tangible long term benefits to offset clear, often apparently high, short term costs can severely hamper the speed and scope of eGovernment progress (Eynon 2007). A key challenge is to find ways of calculating benefits whilst also recognising the true costs and risks. Such calculations must be based on public sector value models. In contrast to the business sector, the public sector has to increase not only economic values but also societal (e.g.

equality and rule of law) and democratic (e.g. openness and transparency) values. Not only does this add to the list of goals to be strived for, there are typically conflicts between goals (Irani et al. 2005). Public sector activities also involve a wide variety of target stakeholders (Flak and Rose 2005, Scholl 2005) and hence require trade-offs not easily decided.

To meet this complex situation there are a number of models developed to measure benefits in the public sector in general and for eGovernment projects in particular. Two such models are one developed by the OECD and one by the EU (Lonti and Woods 2008, Codagnone and Boccardelli 2006). Both models address three categories of benefits, (internal) efficiency, (external) effectiveness, and democracy/openness. While there is no single universally used standard model for measuring values produced in the public sector, the examples given above show that there are at least good candidates.

Measuring benefits is not just a matter of having assessment models. There is also a need for implementing the measurement criteria of such models as structured ways of working. This is commonly called Benefits Management, a term comprised of Benefits Realisation and Change Management (Ward and Daniel 2006). While still an emerging field, there are lessons to be learned from existing experiences with benefits management methodologies. Such approaches generally address issues related to realising and managing benefits in organisations and are thus considered relevant in this context.

It can be argued that also benefits management is too restricted, being focused on the benefits within an individual organisation. All complex eGovernment systems involve interaction of several different public and private services. To achieve higher value levels, it is necessary to depart from pursuing only traditional strategies (efficiency) and aim at value innovation. To accomplish this, collaboration needs to be developed and reinforced. This requires changes not only in the information systems and business processes inside the different organisations, but also calls for process innovation across traditional public and private organisations (Flak et al 2009) as they constitute ecosystems for service delivery.

No matter how value is defined, the overarching world-view that is applied influences the more concrete method of achieving this value. FRISCO (Falkenberg et al. 1996) distinguishes between three different views of the world:

1. 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.

2. Constructivistic: "Reality" exists independently of any observer, but what each person possesses 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
3. 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.

Methodologies can be categorised as being either 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 the underlying Weltanschauung. The distinction into objectivistic and constructivistic above is parallel to the distinction between objectivistic and subjectivistic in the overview of (Hirschheim and Klein 1989) Hirschheim and Klein also distinguish along the order-conflict dimension. In this dimension, the order or integration view emphasises a social world characterised 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 1979) in the context of organisational and social research.

Based on the discussion in Chap. 1, it should come as no surprise that we find it beneficial to adapt a constructivistic world-view. Note however that we have a somewhat different approach to constructivism than the one described in the FRISCO-report.

2.1.2 Coverage in Process

Looking at the introduction of this chapter, we see a differentiation between four major types of tasks in IT support in organisations:

Do the methodology address:

- Development of application systems
- Use of application systems in production
- Operation of application systems in production
- Maintenance and evolution of application systems in production

One or more of the above four 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 1994) and (Davis 1988). 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. The product vs. problem-oriented dimension as discussed by Blum is in our view a distinction on the part of development that is covered. The differentiation in abstraction level (analysis, requirements, design, implementation) is a useful, more detailed breakdown of this. In some methodologies (e.g. agile methodologies), one apparently collapses these abstraction levels, but it is important to differentiate between these as illustrated by (Davis 1995), looking upon the differences between analysis (OOA), requirements (OOR) and design (OOD) when working with object-oriented systems:

- There are different (but probably overlapping) set of objects in OOR and OOD. Is the object important in the problem domain (OOR) vs. can the object be constructed in a way that make efficient use of data abstractions (OOD)
- There are different reasons for not having a System-object: In OOA, the domain do not contain the system, in OOD, the object model is the system.
- There are different reasons for using aggregation. At design time, it is important to record whole-part relationship to achieve optimal packaging.
- The focus on instantiation is different. At analysis time, one does not care about creation and destruction of instances.
- At analysis time you do not focus on the detailed algorithms and behaviour of the object
- Whereas generalisation is used in analysis for improved understanding, in design it aids delegation, productivity and encapsulation
- Verification and validation are done towards different models, in design it is primarily to find that the design satisfy the requirements, whereas in analysis, one needs to find if the requirements satisfy the human and organisational needs.

Thus even if you use the same concepts in analysis and design, your models are fundamentally different since you can say that the domains you model are fundamentally different.

We claim that a comprehensive methodology should cover all the four areas development, use, operations and maintenance in an integrated manner. The emphasis in this book will be put on development and maintenance/evolution, but also the usage aspect is important, enabling the different end-users to make sense of the existing applications system in the organisation, 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 IT-support of the organisation when the environment of the organisation is changing. To have an integrated view of development and operations is important for getting the value out of the IT-investments (Iden et al. 2011).

We also claim that it is beneficial to not differentiate between development and maintenance as distinctly as is done in many approaches.

Maintenance was earlier looked upon as a more boring and less challenging task than development. According to our discussion in the introduction of this chapter, it is both natural and desirable for information systems 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 portfolio, and should be given credit as such. On the other hand, almost half of the new systems being developed are replacement systems, being application portfolio upkeep not enhancing the functional coverage of the portfolio or in other words, what the users can do with the system. 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, and instead focus on the percentage of application portfolio development (Davidsen and Krogstie 2010, Krogstie 1996). This is difficult to achieve when keeping a large mental and organisational gap between development and maintenance, even though the actual tasks being done have many similarities.

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

- A large proportion of traditional maintenance work is to perform reverse engineering 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. Another thing is that also when developing new systems, they have to be integrated with existing systems, thus reverse engineering is also an important activity in these cases. 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 a problem 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, or because developers are transferred to other projects.

- It is generally believed that ``Maintenance of systems is characterised 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). Looking in detail on the distribution of maintenance activities, as reported in (Benestad et al, 2009; Gupta, 2009), it appears to be very large differences reported in different studies. Whereas Lientz and Swanson (1980) reported 60% perfective, 18% adaptive and 17% corrective maintenance when asking about selected systems from a large number of organisations (one per organisation), Sousa and Moreira (1998) reported (based on a number of systems in one organisation) 49% adaptive, 36% corrective, and 14 % perfective maintenance. (Mohagheghi and Conradi, 2004) reported 53% corrective, 36% perfective and 4% adaptive maintenance, based on data on three open source products. (Lee and Jefferson, 2005) reported 62% perfective, 32% corrective, and 6% adaptive maintenance based on data from one application in production. Fig. 2.1 summarises the results from our own investigations where we look upon the complete portfolio of a number of organisations (i.e. a more aggregated view that is less influenced on where the system is its lifecycle). Most interesting for comparison with other surveys is looking at corrective, adaptive, and perfective maintenance, which appears to be much more stable than the numbers reported from others above. (Jørgensen 1994) indicates that the assessed corrective percentage of the work used on maintenance often might be exaggerated since these kinds 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 investigations, 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.

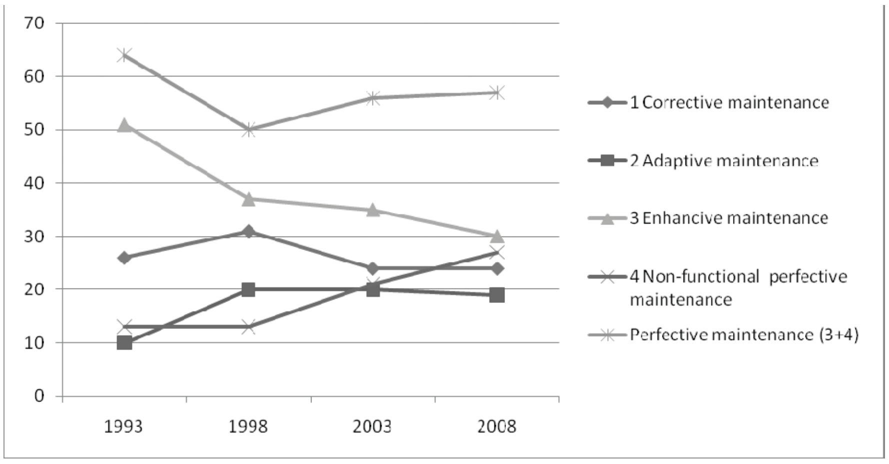


Fig. 2.1 Comparisons of distribution on maintenance tasks, percentage

The effect of this would then probably decrease the total amount of functional maintenance even further, thus indicating an even better situation regarding average functional maintenance. Related to this are the results of a survey reported on in (Dekleva 1992) which gave no conclusive evidence that organisations 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.

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 1994), where 38% of the tasks were of a corrective nature, as much as 2/3 of the tasks where classified to have high importance by the maintainers themselves.

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 artefacts in production, but it has been shown to be possible to reduce this by good development practice (Krogstie 2000). Another aspect here is that with shorter turnaround time and agile development, the visibility of development changes are increasing.

2.1.3 Coverage of Product

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

- One single application system
- A family of application systems
- The whole portfolio of application systems in an organisation
- The whole organisation (developing and maintaining so-called enterprise architectures)
- A cluster of cooperating organisations in combined digital content and software ecosystems (Krogstie 2011)

We can argue that it is beneficial for a methodology to be able to consider the whole portfolio and not only the single application system, although being able to concentrate on different areas at different times.

For the end-users of information systems, it is not important which concrete application system that is changed. What is important is that their perceived needs are supported by the complete portfolio of available systems.

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. They need to adhere to de facto and de jure standards as the applications in different companies are more and more integrated, utilising common resources and components. On national level countries such as Norway is developing common national components (Aagesen et al. 2011).

In our investigation from 2008 (Davidsen and Krogstie 2010)), 94 new systems were currently being developed, and as many as 60 of these systems (64 %) were regarded as replacement systems. (2003-60%; 1998-57%; 1993-48%). The portfolio of the organisations responding to this question contained 446 systems, meaning that 13% of the current portfolio was being replaced. (2003-13%; 1998-9%; 1993-11%). The average age of systems to be replaced was 7 years (2003-5.5 years; 1998-7.7 years; 1993-8.5 years).

More detailed reasons for the replacements are reported in (Davidsen and Krogstie 2010). Here one sees that the main motivation for replacement has changed slightly from earlier investigations. The most important reasons for replacement in the 2008 investigations are partly a need for integration and burden to maintain and operate which appears to become more important again, a bit surprising giving the relatively young age of the systems that are replaced. On the other hand, the web-systems developed over the last 10 years relates to much more demanding and volatile operating environments than the primarily in-house systems developed earlier.

Often when doing system integration, it can be useful to collect the functionality of several existing application systems into a new application system, something that is not well supported when having strict borders for what is inside and outside of an application system.

As noted already in (Swanson and Beath 1989) the CISs of an organisation 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 organisations surveyed were dependent 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 organisation is served more by its CISs as a whole than it is by the application systems taken individually.

In addition to the core systems used by the organisation, a number of supporting systems are used to develop these and operate these. A normal differentiation is between execution architecture (what applications run on), operating architecture (systems needed in addition for efficient operation and user-support) and development architecture (systems used to develop, evolve and maintain all the other systems).

2.1.4 Capabilities for CIS-Portfolio Evolution

For the different tasks within a methodology, and for the different products as part of the systems portfolio, a number of skills need to be available on the personal and organisational level to ensure that value is achieved.

A good source in relation to development is the person-roles described in Rational Unified Process (RUP) (Kruchten 2003), although the descriptions are written specific to the development of object-oriented systems, RUP role definitions are consistent with the notion of separating breadth and depth. Personality types for breadth workers and depth workers are very different. Breadth work is quick, inexact, and resilient. Depth work takes much more time, requires attention to detail, and must be of significantly better (syntactic) quality. Each of RUP's disciplines has one role that focuses on breadth for that discipline, and a different role that focuses on depth for the same discipline. Table 2.3 lists each RUP discipline along with its corresponding breadth and depth roles, and briefly explains the roles' functions, being generalised to be applicable also in other methodologies than RUP.

Table 2.3 Breadth and depth roles relative to methodology disciplines

Discipline	Breadth role	Depth role
Analysis	Business Analyst Discovers and analyse all business requirements.	Business Designer Details a single set of business requirements.
Requirements	Systems Analyst Discovers all system requirements.	Requirements Specifier Details a single set of system requirement.
Design	Software Architect Decides on technologies for the whole solution.	Designer Details the design for a part of the solution (e.g. in RUP relative to a single use case).
Implementation	Integrator Owns the build plan that shows what system components will integrate with one another.	Implementer Codes a single set of system components
Test	Test Manager Ensures that testing is complete and conducted for the right motivators.	Test Designer Implements automated portions of the test design for the iteration.
	Test Analyst Selects what to test based on the motivators.	Tester Runs a specific test.
	Test Designer Decides what tests should be automated vs. manual and creates automations.	
Deployment	Deployment Manager Oversees deployment for all deployment units.	Tech Writer, Course Developer, Graphic Artist Create detailed materials to ensure a successful deployment.

Project Management	Project Manager Creates the business case and a coarse-grained plan; makes go / no go decisions.	Project Manager Plans, tracks, and manages risk for a single iteration. (Note that this discipline has only one role. Assigning the depth view to a project coordinator can provide relief for project managers.)
Environment	Process Engineer Owns the process for the project.	Tool Specialist Creates guidelines for using a specific tool.
Configuration and Change Mgt	Configuration Manager Sets up the configuration management environment, policies, and plan.	Configuration Manager Creates a deployment unit, reports on configuration status, performs audits, and so forth.
	Change Control Manager Establishes a change control process.	Change Control Manager Reviews and manages change requests.

In simpler methods, such as Scrum (Schwaber and Beedle 2002), the number of roles are smaller. Scrum also illustrates the concept of organisational role, and the related capability of the role. Scrum has three roles: product owner, ScrumMaster, and team (consisting of individual team members).

Product Owner: The product owner decides what will be built and in which order.

- Defines the features of the product or desired outcomes of the project
- Chooses release date and content
- Ensures profitability (ROI)
- Prioritises features/outcomes
- Adjusts features/outcomes and priority as needed
- Accepts or rejects work results.
- Facilitates scrum planning ceremony.

Scrum Master: The Scrum Master is a facilitative team leader who ensures that the team adheres to its chosen process and removes blocking issues.

- Ensures that the team is fully functional and productive
- Enables close cooperation across all roles and functions
- Removes barriers
- Shields the team from external interferences

- Ensures that the process is followed, including issuing invitations to daily scrums, sprint reviews, and sprint planning.
- Facilitates the daily scrums

The team:

- Is cross-functional
- Is right-sized (the ideal size is seven (plus/minus two) members);
- Selects the sprint goal and specifies work results;
- Has the right to do everything within the boundaries of the project guidelines to reach the sprint goal;
- Organises itself and its work; and
- Demonstrates work results to the product owner and any other interested parties.

When extending the coverage of the product to be the whole organisation, not only the IT-systems, the need for participatory enterprise modelling and enterprise architecture skills and knowledge get more pronounced. The following lists the needed competences in this regard (Persson and Stirna 2010) distinguishing between modelling competence and management competence.

Competences related to modelling

- ability to model, i.e. making use of the chosen modelling language to create and refine enterprise models. Knowing how to use modelling tools for documenting and analysing the modelling result is also included in this capability
- ability to facilitate a modelling session: This capability is very much based on knowledge about the effects of modelling, the principles of human communication and socialisation (especially in groups), as well as the conditions of human learning and problem solving (cognition).

Competences related to managing enterprise modelling projects

- ability to select an appropriate EM approach and tailor it in order to fit the situation at hand.
- ability to interview involved domain experts
- ability to define a relevant problem.
- ability to define requirements on the results
- ability to establish a modelling project
- ability to adjust a presentation of project results

- ability to navigate between the wishes of various stakeholders
- ability to assess the impact of the modelling result and the modelling process

We will return to how these skills are used in different modelling tasks in the section on participatory modelling in section 2.4.

2.1.5 Stakeholder Participation

A large number of persons playing different roles in connection to IS-development and evolution. In general, stakeholders in information systems implementation 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. These are typically termed the 'customer' of the system.
- Those who have an interest in its use, for example direct or indirect users and user's managers. Direct users are often called end-users.

In relation to the last group in particular, different methods differentiate between how they take part activities related to making or evolving the information system. The term 'participation' means to take part in something, most typically to take part in and influence a change of some sort that again influences the different stakeholders of the change. The change is typically done in some organisational or societal context, where someone uses resources to perform the change, to achieve some sort of value (cf. section 2.1.1 above).

There exist different forms of participation:

- Direct participation: Every stakeholder has an opportunity to participate.
- Indirect participation: Every stakeholder participates 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 peers

Based on the discussion in (Mumford 1983, Mumford 1986) there are a number of different reasons for participation, partly based on the power position of the stakeholders.

1. Morally right: The classical reasoning behind participation, based on a vision of a common good (e.g. justice and freedom) and how universal involvement in decision taking could help ensure this.
2. Educational: Involvement might provide understanding and knowledge among those participating that can assist an organisation to more effectively realise its objectives.
3. Improved ownership/motivation among stakeholders, resulting in better effect of the change. Focus on that by being involved in the process the stakeholders will be more willing to take the changes produced in use.
4. Leveraging of power. Unions for instance may encourage participation because they see it as a lever for increasing workers control over the work situation and contribute to industrial democracy.
5. Protect against unwanted solutions: In a company, employees might see participation as a way to prevent things that they believe to be undesirable from happening (i.e. being made redundant, or deskilled).
6. Transparency: Ensure that the decision-making process behind the changed solution is (perceived to be) open and trustworthy (i.e. not based on hidden agendas).
7. Emancipation: Enables those participating to feel free, be their own masters, and in control of their own destinies.
8. Character-building: It assists people to develop active, non-servile characters and democratic personalities, and also enables them to broaden their horizons and appreciate the viewpoints and perspectives of others.
9. Improved solution. Through participation, more relevant knowledge is available and thus a better solution can be provided.

Whereas the above indicate positive reasons for participations, there might also be negative reasons, e.g. to involve people as 'hostages', persuading people to accept changes that otherwise might be rejected, or having possible scapegoats if a change is not successful.

Based on the reason for participation, the value to be achieved from participation is obviously varying. Often in a systems development setting, the primary reasoning for having participation is the last in the above list(to have an improved solution), but you can also have variants of the others by ensuring stakeholder *influence*. A key aspect of the influence principle is to view "users" as active and competent partners and domain experts. Equally important is to base these innovations on the needs and desires of potential users, and to realise that these users often represent a heterogeneous group.

In information systems literature user participation is basically motivated applying a cost-benefit-perspective through achieving a better solution. 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 organisation. Added to this is the general argument of including those who are believed to have relevant knowledge in the area, and which are influenced by the solution.

We thus focus here specifically on participation of (potential) end-users of the information system. Note that a lot of the user involvement literature, related to the so-called Scandinavian School and Participatory Design (Bjerknes and Bratteteig 1995, Schuler and Namioka 1993) have a different outset with a focus on democratic participation, where the empowerment of employees through participation is often regarded a goal in itself.

According to (Heller:91), participation is sharing power and influence. One can classify the influence by the stakeholder of a change on the following levels (Arnstein, 1969;Heller, 1991)

0. Manipulation by giving wrong or biased information
1. No information to stakeholders
2. Information to stakeholders provided
3. Consultation with stakeholders (e.g. hearings)
4. Advice of stakeholders taken into account
5. Common decision between stakeholder (problem owners) and developers
6. Delegated authority to stakeholders
7. The problem-owners have full control

Note that the higher on this scale one are, the *responsibilities* laid on the stakeholders also increases.

Due to the large number of potential stakeholders in IS development and evolution, in most cases representative participation will be the only practical possibility. From the point of view of social construction (Berger and Luckmann 1966), it is doubtful that a user representative can truly represent anyone else than himself. Different perspectives and views on the reality are also often mentioned reasons for why it is crucial to involve users as well as many different types of stakeholders in the development process. The reality aspect is also considered by focusing on involving real users, not basing on personas (Grudin and Pruitt 2002) or other user representative theories. 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 (Orlikowski 1994).

Another factor is the scope of participation, i.e. when does participation take place. Usually one would expect that user-participation would take place heavily in analysis/requirements and in acceptance testing, more lightly in technical design, and very little in implementation, but this will often depend on the chosen methodology. For instance in agile development (Abrahamsson et al. 2010) a main point is to try to have user participation also in the implementation, to ensure short turnaround time from changes in the technical solutions to feedback on these changes.

When it comes to suggesting improvements and evolutions of the current information system of the organisation, direct participation should be made possible to a larger degree, e.g. illustrated in the use of Active Knowledge Modelling (AKM). Also in the project establishment, a larger proportion of the stakeholders should be able to participate.

Practices and methods of user involvement include a multitude of different traditions and approaches. One useful way to analyse this multitude relative to for instance governmental systems is to differentiate between (1) methods and practices designed to fit information systems development processes and (2) government participation practices (Følstad et al. 2004).

The field of Human-Computer Interaction (HCI) is an example of an approach developed to conduct user involvement in software development projects. This field covers methods and practices suited for the development phases of analyses, requirement specification, design, and evaluation. HCI methods for analyses include analyses of users and stakeholders, user tasks and context of use; methods for requirements elicitation and description include work-shops, interviews, field studies, personas, and use cases; methods for design include rapid prototyping, design patterns, card sorting, and story boarding; and methods for evaluation include analytical methods like cognitive walkthrough and heuristic evaluation as well as empirical methods like user tests and field evaluations (Maguire, 2001).

Within in particular the governmental sector a different form of user involvement has evolved. Government user involvement practices typically include: User reference groups following the development project for a substantial period of time, inclusion of user representatives in the project team, inclusion of user and stakeholder representatives in project steering committees, formal and semi-formal audits of project plans and system specifications, workshops with user and stakeholder representatives, involvement of user interest organisations, public meetings, and public information activities.

2.1.6 Organisation of Development and Evolution of Information Systems

A number of organisations can be involved in IS development and evolution, and these can be organised in different manners

Early on IT was interleaved with other activities in the organisation, but as the importance of IT increased, a separate organisation appeared, taking care of development, maintenance (application management), systems operation, user support and management. Slowly, this has been divided in distinct organisational units. Actually, since one always has based internal IT-systems on existing technology, one has had to deal with external vendors of equipment and systems. As more and more packaged systems has appeared, both with large and small adaptations, the number of external partners developing and maintaining the information system support of organisations have gradually increased. The increasing use of open source, and the new development of governmental common components (Aagesen et al. 2011) illustrates this development. For instance within OSS (Fitzgerald, 2001) hundred thousands of co-developed software components are freely available, often via portals like http://sourceForge.net. The quality is variable and often poorly documented.

The following illustrates one way of dividing the tasks in different organisational units from a Norwegian organisation after outsourcing parts of the tasks, primarily to organisation NN (Krogstie 2000).

Application management (AM) includes the development of new releases of existing systems, error corrections and handling change requests to the existing application portfolio of the organisation. New releases are typically developed through small projects, having all the normal tasks of a system development projects such as analysis, requirement specification, design, programming, testing, deployment, and project management. IT-operations were out-sourced to a third company, and also several other companies were responsible for the maintenance/evolution and further development of selected packaged systems. The IT-steering model in the organisation is illustrated in Fig. 2.2. The company retained some people in the IT-function to have the overall control of the IT-strategy and vendor agreements, and to ensure sufficient in-house knowledge to use IR in a strategic manner. The different company divisions and steering groups were decision-makers on overall IT-spending and project-direction. Two coordination bodies were established. Program management, coordinating dependencies across all larger ongoing projects and application management releases, and Service Management, controlling the overall IT-service levels, SLAs, requests, errors etc. The system developers in the AM unit were divided into 7 groups, each supporting a separate part of the highly interrelated application portfolio supporting the activities of organisation.

Earlier some work was done to investigate the pros and cons of differentiating development and maintenance activities. Already (Chapin 1987) investigated into this area and concluded that no differences in the nature and characteristics of the demand for or performance of software maintenance work was found between the two types of departmentalisation. What those organisations adopting a separate organisational place for software maintenance had achieved was fewer management problems and a more positive attitude toward software maintenance by those managing it, something we believe that also can be achieved by not differentiating between development and maintenance projects in the first place.

As for system operations, it is more usual to have this in a separate organisation (Iden et al. 2011) than systems development and evolution.

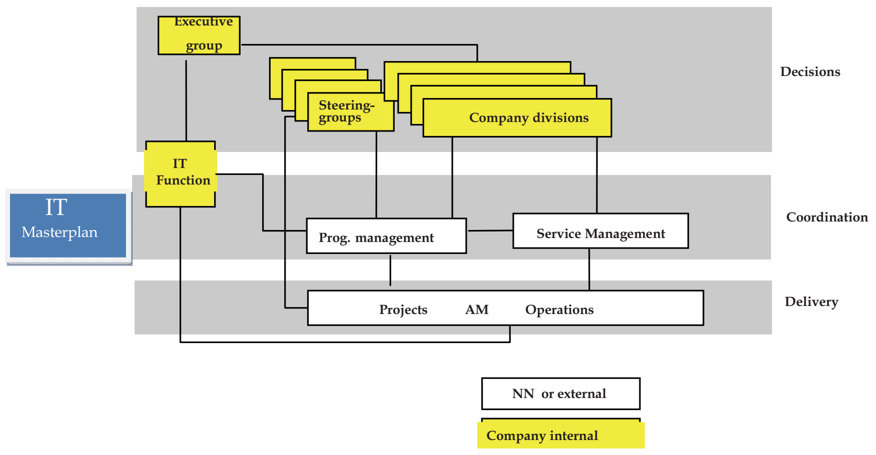


Fig. 2.2 Example of an IT-Steering Model

2.1.7 Location: Where the Work Take Place

This is often looked upon in connection with the organisation, but is actually another dimension. A lot of the focus of agile development is that people involved in the development project should be co-located, no matter if they actually worked in the same organisation in different parts of the organisations, or were external consultants. Similarly in the model presented in Fig 2.2 most of the workers in the different organisational units were actually co-located even if working for different organisations.

One example of a notable change from our survey investigations mentioned in the introduction of this chapter is where systems are developed, maintained and operate compared to 15 years ago. In 1993, 58% of the systems were developed by the IS-organisation, and only one percent was developed *in* the user organisation. In 1998, however, 27% of the systems were developed by the IS-organisation and 27% as custom systems *in* the user organisation. In 2003, 23 % of the systems are developed in the IS-organisation, whereas in 2008 only 12% was developed in the IS organisation. The percentage of systems developed by outside firms is higher in 2008 (40% vs. 35% in 2003, vs. 22% in 1998 vs. 12 % in 1993 vs. 15 % in Swanson/Beath). The percentage of systems developed based on packages with small or large adjustments is also comparatively high (41% in 2008 vs. 39% in 2003 vs. 24% in 1998 vs. 28% in 1993 vs. 2% in Swanson/Beath). The new category we introduced in 1998, component-based development (renamed "use of external web services" in 2008) is still small (5% in 2008) although increasing (1.0 % in 2003, 0.4% in 1998).

Whereas earlier the main way of developing systems where to do it in-house or using temporary consultants (potentially based on adapting packages), outsourcing of all type of IT-activities has been on the rise over the last 10 years, although you also find examples lately on in-sourcing (were earlier outsourced tasks that are taken back into the organisation). In the 2008 investigation around a third of the IT-activity was outsourced (32.9% in private sector, 24.1 in public sector (Krogstie 2010)). Whereas only two of the respondents reported to have outsourced all the IT-activities, as many as 84% of the organisations had outsourced parts of their IT-activity. Whereas the public organisations have outsourced more of the development (40% in public, 29% in private) and maintenance (34% in public, 30% in private) work than the private organisations, they have outsourced less of the operations (31% in public, 41% in private) and user support (21% in public, 29% in private). Other important aspects to take into account are the rise of new delivery models (e.g. OSS) not only for infrastructure applications, but also business application. It is reported that 50%

of Norwegian IT companies now integrate OSS components into their own applications, and 10% also contribute back to the OSS community (Hauge et al. 2010).

Another trend is towards using cloud infrastructures for the operations of IT-solutions.

The conceptual idea underlying Cloud Computing started its development as early as the 1960s when John McCarthy expressed that "computation may someday be organised as a public utility". This thought was further developed in (Parkhill 1966) where he compared the supply of computational power to the supply of electrical power. This analogy is oversimplified, i.e. the data transferred over the network represent symbol structures, electrons in electric power do not, but it gives a basic idea of the concept. Cloud Computing is relatively new term within the IT-field. National Institute of Standards and Technology (NIST) defines Cloud Computing as (Mell and Grance 2009):

> A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The NIST definition describes five essential characteristics of cloud computing. On-demand self-service, broad network access, resource pooling, rapid elasticity and measured service.

1. On-demand self-service: Clouds offer automatic scaling of resources without human interaction.
2. Broad network access: Applications hosted in the cloud are often available from thin-clients like mobile devices and web browsers thereby offering great flexibility in the choice of end-user device.
3. Resource pooling: The cloud uses virtualisation of resources and therefore appears as one large pool of resources. These resources are dynamically assigned to the users of the cloud. The users of cloud services generally does not know the exact location of the resources being provided, but can in some cases define a location at a higher level of abstraction (e.g. country, continent).
4. Rapid elasticity: Demanded resources in the cloud can be quickly allocated to the customer. The cloud's resources often appears as unlimited from the customers point of view.
5. Measured service: The system automatically monitors and controls the resources used by each customer. The most used method of payment is

a pay-per-use model where the customer pay for the resources used. This requires monitoring and control of resources.

The NIST definition describes three different service models. The three levels offer different levels of abstraction.

1. Cloud Software as a Service (SaaS): A customer can rent an application running in the cloud. The application is often accessed via thin clients like web browsers. The customer does not control the underlying cloud structure and not even the application capabilities except for limited configuration settings for the application. Examples of SaaS providers are Salesforce.com and Service-now.com. The model has been available for quite some time, and was earlier marketed under the term ASP (Application Service Provider).
2. Cloud Platform as a Service (PaaS): The customer is offered a platform where customer created applications can be deployed using APIs offered by and programming languages supported by the service provider. The customer does not control the underlying structure (i.e. operating system, network or storage). An example of this is Microsoft's Windows Azure platform.
3. Cloud Infrastructure as a Service (IaaS): The cloud infrastructure, usually a platform virtualisation environment, is delivered as a service. The customer can control a large part of the platform (i.e. operating system and storage).

Customers using cloud computing have different needs when it comes to data privacy and data security. Some have data that they will not risk being accessed by others, some organisations share privacy concerns and can use the same cloud (e.g. different governmental agencies needing to store data on the citizens), while others do not have any concerns with sharing computational resources with others. The NIST definition of cloud computing describes four deployment methods for clouds. A deployment method describes who has access to the cloud.

1. Private Cloud: A private cloud is as the name says; private. Private clouds are typical for organisations that have sensitive data that they do not want to risk unauthorised access to, but still need the other characteristics of cloud computing.
2. Community Cloud: A community cloud is a private cloud for a larger group of organisations. This may be organisations that require the same type of security. They may also use the same type of data in their applications.

3. Public Cloud: A public cloud is a cloud infrastructure that is available to everyone (the public). Applications and data can coexist independent
4. Hybrid Cloud: A hybrid cloud is a combination of the previously mentioned cloud types. An organisation can run a private or community cloud and utilise the resources of a public cloud if it is needed for restricted parts of their applications.

For organisations as large as national governments a private cloud does not differ very much from community cloud.

As providers of software through open source and mash-ups get's more professional, one has started to talk about software ecosystems (Messerschmitt and Szyperski 2003). A *software ecosystem* (Jansen et al. 2009) is

> ”a set of businesses functioning as a unit and interacting with a shared market for software and services, together with relationships among them. These relationships are frequently underpinned by a common technological platform and operate through the exchange of information, resources, and artefacts”

We will return to software ecosystems in and other new organisational forms chapter 8.

2.2 A Short History of IS Methodologies

Although we find a large number of methodologies, we can recognise back to the work in the sixties (Langefors 1967), a differentiation according to abstractions-levels. Crudely, as discussed above in Sect. 2.1.2, the following abstraction levels are found in most methodologies:

- analysis
- requirements specification
- design
- implemented systems

A main differentiation between different methodologies is how we organise the work on the different abstraction levels , relative to scope, time, and persons involved, and iterations between these (Berente and Lyytinen 2007). In the previous section, we presented a framework for IS methodologies meant to clarify facets of this. Here we will briefly describe some important methodology developments. Particular model-based approaches are discussed in section 2.3.

There is a common understanding that one in the early years of computing (50ties and 60ties), was basically following a naive approach of code-and-test. Actually already in the sixties, a number of theoretical notions differentiating the different levels of abstractions for information systems were described by Langefors (Bubenko 2007). They appeared when Langefors was at the Swedish SAAB aircraft company (e.g. (Langefors 1963)). In 1967 many of these reports were compiled in (Langefors 1967).

Langefors introduced a number of concepts related to modelling for information systems among others the partitioning of the system development life-cycle in four important *method areas*

- Methods for management and control of organisations
- Methods for analysis and description of information systems at an elementary, “problem oriented” level (the “infological” realm)
- Methods for design and analysis of computerised information systems on a "product-oriented" level (the “datalogical” realm)
- Methods for implementation of the information system on computer hardware and choice of hardware.

2.2.1 The Waterfall Methodology

An important contribution of the sixties was the confirmation of the significance of the infological realm, i.e. the realm where data processing problems were expressed formally, but in a machine-independent way. This laid the basis for a number of new modelling notions during the next decade. The differentiation also made its way into mainstream systems development methodology, first in the form of the so-called waterfall model. The waterfall model, illustrated in Fig. 2.3, is usually attributed to (Royce 1970)

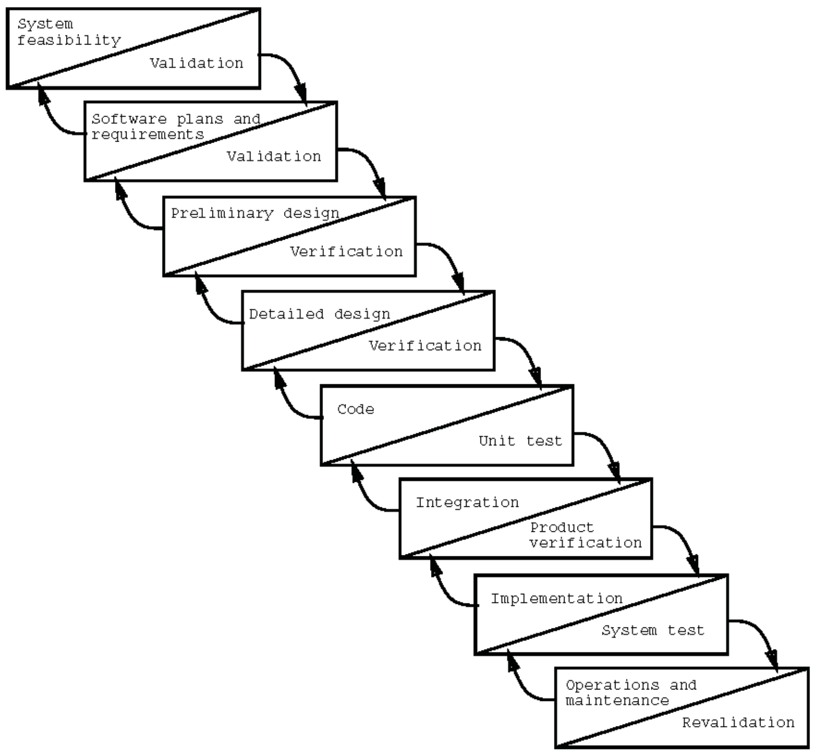


Fig. 2.3 An example of steps in a waterfall methodology

The waterfall methodology organises CIS projects as a linear sequence of phases, where each phase is completed by documenting its achievements. In addition, there are feedback loops between successive phases that enable the modification of the documents from the previous phase.

The perceived benefits of using the waterfall model can be summarised as follows (Davis et al. 1988b):

- It instructs the developers to specify the system prior to the construction of it.
- It encourages one to design the system components before they are actually coded.
- By viewing the project as a sequence of phases, the managers can more easily control the progress of the work, and use the defined milestones as a tool for deciding if the project is to continue or not. It will also help the managers to set up a structured and manageable project-organisation.

- One is required to write documents that will ease the testing and documentation of the system, and that will reduce maintenance costs.

Every project starts out with a feasibility study. The main problems are identified and the pursuit of a new CIS is justified in terms of unfulfilled needs and wishes in the organisation to be supported by the information system.

The purpose of the requirements specification is to define and document all the stakeholder's needs. The specification is supposed to contain a complete description of what the system will do from a user's point of view. How the system will do it, is deliberately ignored. This is meant to be addressed during design.

Most early ``standard'' methodologies for commercial organisations, government contractors, and governmental organisations followed some basic variation of this model, even if the number of phases and the names of the phases often varies (Davis 1988a).

In was soon realised that the traditional depiction of the waterfall could be improved by linking the test activities (on different abstraction levels) closer to the 'downstream' development levels. The V-model (Beizer 1990) (as shown in Fig. 2.4 builds a logical V shape sequence where the system and acceptance testing are tightly associated with the requirements.

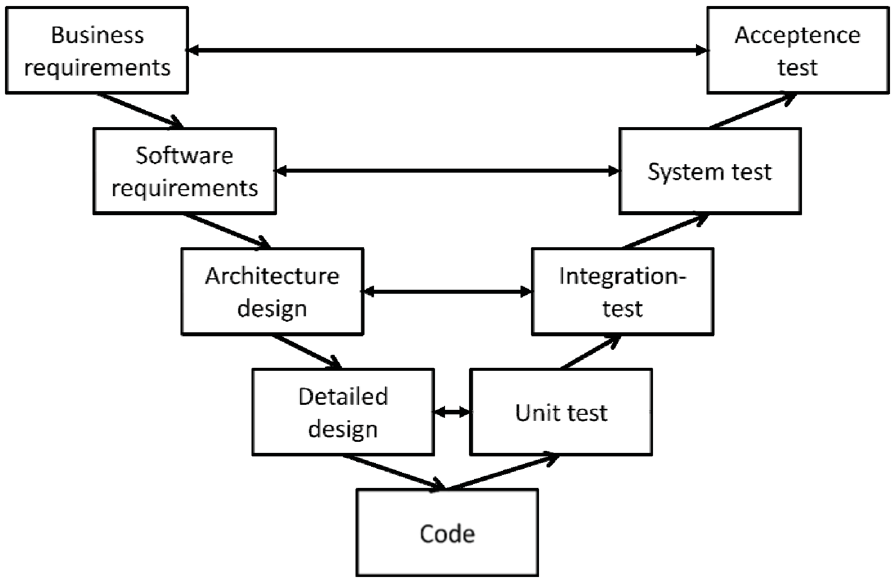


Fig. 2.4 The V-Model relating development and testing activities

The (conventional) waterfall model received much criticism already in the beginning of the eighties (McCracken and Jackson 1982, Swartout and Balzer 1982):

- The phases are artificial constructs, ``one specific kind of project management strategy imposed on software development"(McCracken and Jackson 1982). It is in practice often difficult to separate specification completely from design and implementation.
- An executing system is presented first at the end of the project. This is unfortunate of several reasons:
    - Errors made in the specification will be more difficult and thus more costly to remove when they are not discovered before the end of the project.
    - The customer and end-users may get impatient and press for premature results or lose interest in the project because they do not see any result of the work that has been done.
    - The communication gap between the users and the developers arising because of their different realities not being attacked.

    - Systems developed using the conventional methodology is often difficult to change, resulting in poor support for system evolution.

Similar critiques has been raised in the last decade in connection to the introduction of so-called Agile development methods (Abrahamsson 2010, Beck 1999), but has been attempted to be attacked by a number of works already in the eighties, classical work on alternative approaches which we will discuss first.

2.2.2 Prototyping

A prototype can be defined as ``an executable model of (parts of) an information system, which emphasises specific aspects of that system" (Vonk 1990).

Prototyping as a *technique* is usually seen as a supplement to conventional application system development methodologies (Taylor and Standish 1982, Vonk 1990). It put emphasis on high user participation and tangible representations of selected user requirements at an early stage. The iterative generation and validation of executable models makes the approach particularly useful when the requirements are unstable or uncertain. Another usage is a technically oriented proof-of-concept prototyping, which focus on making sure that the attempted approach is technically possible at all.

Prototyping as *methodology* is a highly iterative process, which is characterised by extensive use of prototypes (Carey 1990, Vonk 1990). The objective is to clarify certain characteristics of an application system by constructing an artefact that can be executed. On the basis of user feedback the prototype is revised and new knowledge and new insights are gained. After a series of revisions, the prototype is acceptable to the users, which indicates that it reflects the user requirements in some specific aspects.

According to (Vonk 1990), prototyping differs from a traditional (waterfall) methodology on the following areas:

- The users can validate the requirements by testing a corresponding executable model. The communication between users and developers is improved in that users can directly experience the consequences of the specified requirements.
- Traditional tool-support for these methodologies has been unable to focus on the user interface aspects. Prototypes exploit the execution of (simplified) user interfaces to improve the externalisation of user requirements.
- The requirements tend to change as the project is carried out. Instability of functional requirements is easily handled through the interactive generation and validation of functional prototypes.

In Vonk's opinion, its main benefit is the reduction of uncertainty. The choice of development strategy, thus, should be based on the judgement of project uncertainty.

Carey (1990) sees the following advantages with prototyping: Faster development time, easier end-use and learning, less labor to develop systems, decreased backlogs, and enhanced user/analyst communication. Some disadvantages include: The fostering of undue expectations on the part of the users, what the users sees may not be what the users gets, and the availability of application generator software may encourage unduly end-user computing.

There are two main types of prototyping as illustrated in Fig 2.5: In iterative prototyping, also called evolutionary prototyping, the prototype evolves into the final application system after a series of evolutionary user-initiated changes. In throwaway prototyping the prototype is only used to help to establish the user's requirements to the application system. As soon as the process is finished, the prototype is discarded and the real application system is implemented.

Another classification of prototypes is based on the particular aspects that are included i.e. the focus.

- Functional prototypes include some functionality of the system, and will often contain simple algorithms and data structures.
- User interface prototypes are used to design both the presentational and the behavioural part of human-computer interfaces, simulating the core functionality of the system.
- Performance prototypes concentrate on workload and hardware characteristics.

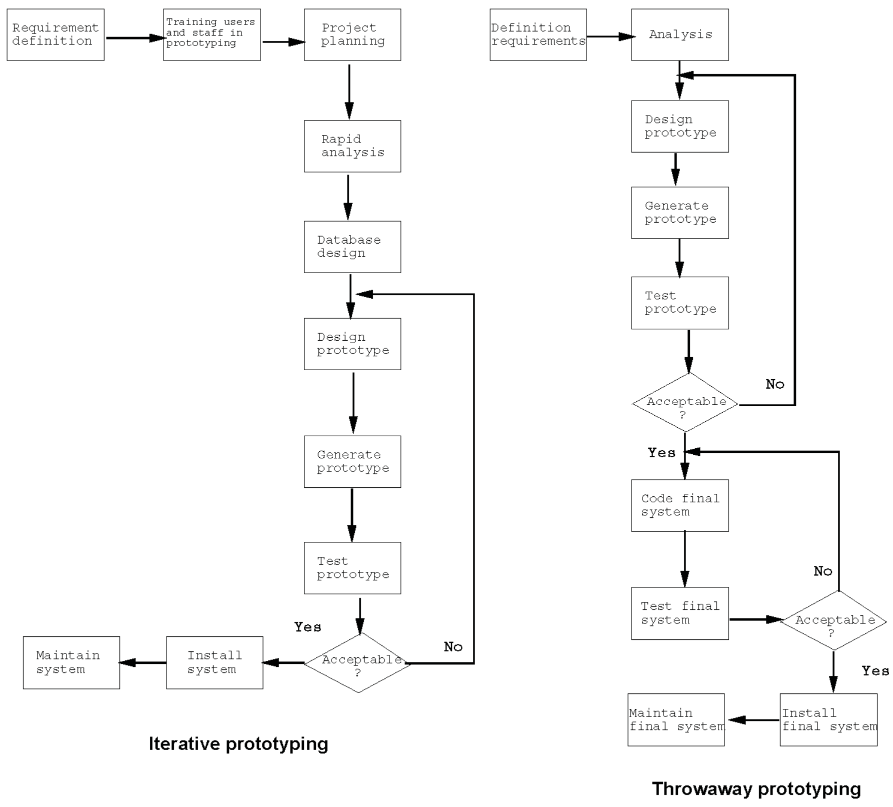


Fig. 2.5 Different forms of prototyping methodologies

2.2.3 Transformational and Operational development

The transformational approach which was originally also often termed automatic software synthesis (Lowry and McCartney 1991) assumes the existence of formal specification languages and tools for automatic transformations.

Its main philosophy, gradually transforming formal requirements specifications into target code, has proved to be a very ambitious goal, but still a goal-pursued in current approaches such as MDA (being described in Sect. 2.3.2).

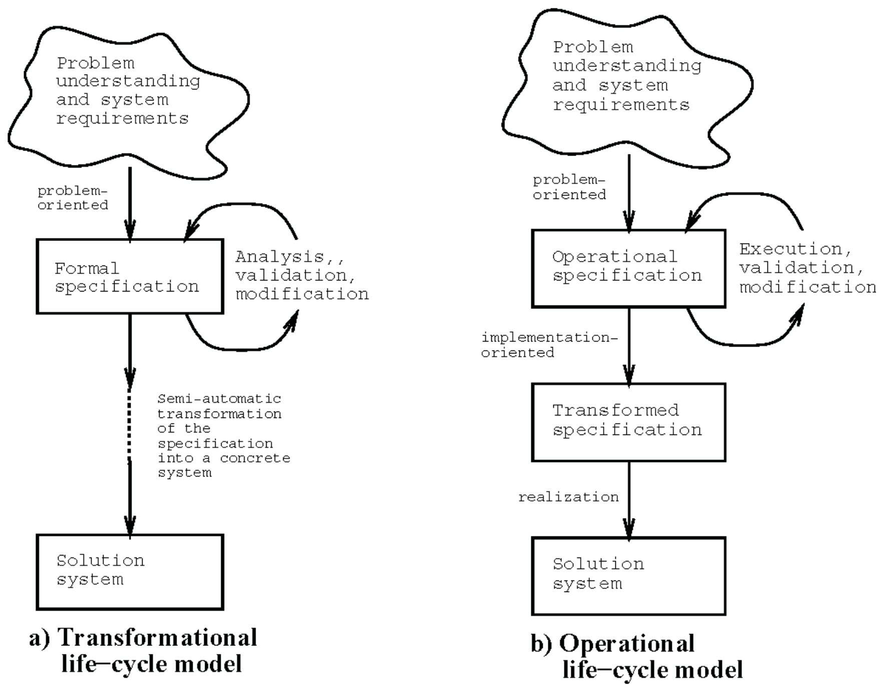

Problem understanding and system requirements
problem-oriented
Formal specification
Analysis,, validation, modification
Semi-automatic transformation of the specification into a concrete system
Solution system
a) Transformational life–cycle model
Problem understanding and system requirements
problem-oriented
Operational specification
Execution, validation, modification
implementation-oriented
Transformed specification
realization
Solution system
b) Operational life–cycle model

Fig. 2.6 The transformational and operational life cycle models

As shown in Fig. 2.6a, the formal specification is the crucial element. It is used as a starting point for the transformation process, but is at the same time the main document for stakeholder validation. This illustrates the main problem using transformational techniques: On the one hand formality is necessary to apply automatic transformations, on the other hand the formality normally makes specifications more difficult to understand for end-users.

A series of transformations are applied to reach the final target code. During these transformations, additional details are added to the specification. Not all of these details can be automatically added, and a developer is needed to guide the transformation process.

The sequence of transformations and decisions is kept in a development record. Using this record, one can maintain and re-implement the formal specification.

The operational approach was first described in detail already by (Zave 1984). Its main characteristics are (1) the separation of problem-oriented and implementation-oriented system features, and (2) the provision of executable system models early in the development process. It is claimed that the approach will enhance the validation process as prototypes are immediately available. The approach is illustrated in Fig 2.6b.

The approach rests on the use of an operational specification language. This language is defined as a suitable interpreter making the models available through prototyping or symbolic evaluation or both (Harel 1992) for validation.

The specification model is rarely suited as the final program code. It is a functional model, although ideas to include non-functional requirements have been explored. Resource management and resource allocation strategies are usually omitted, and characteristics of the target environment are deliberately ignored. As stated by (Agresti 1986), the operational paradigm violates the traditional distinction between ``what'' and ``how'' considerations. Instead the development process is separated on the basis of problem-oriented versus product-oriented concerns.

As soon as the operational model is finished, a series of transformations are carried out. The goal of these transformations is to reach another specification, which is directly interpretable by the target processor. In order to do so, decisions concerning performance and implementation resources are made.

The approach is claimed to have several advantages to conventional waterfall life cycle models (Zave 1982)

- It exploits the advantages of formality (e.g. for formal analysis).
- Rapid prototypes or symbolically executable units are available all away from the start.
- Since the transformations preserve correctness, it is not necessary to verify the final code.
- All functional modifications are done at the specification level.

In addition, in Zave's opinion the separation of problem-oriented and implementation-oriented issues is useful to improve the system's maintainability. Operational specifications are constructed with maintenance in mind, while transformations try to take care of requirements concerning performance and efficiency. Conventional techniques, on the other hand, support only one decomposition principle. The conflicting issue of efficiency and ease of maintenance must be addressed in the same process, which tends to result in more or less unconscious compromises

Among the disadvantages of the operational approach are the danger for premature design decisions, the difficulties of comprehending the formal specifications for end-user, and the problems of guiding the transformations. An early approach in this area based on the use of conceptual modelling techniques, TEMPORA, is described in further detail in chapter 3.3.5.

2.2.4 The Spiral Model

The spiral model was introduced by (Boehm 1988) . It may be perceived as a framework for systems development, in which risk analysis governs the choice of more specific methodologies as the project progresses. The spiral model potentially subsumes both the prototyping (both iterative and throwaway), operational/transformational, and waterfall methodology.

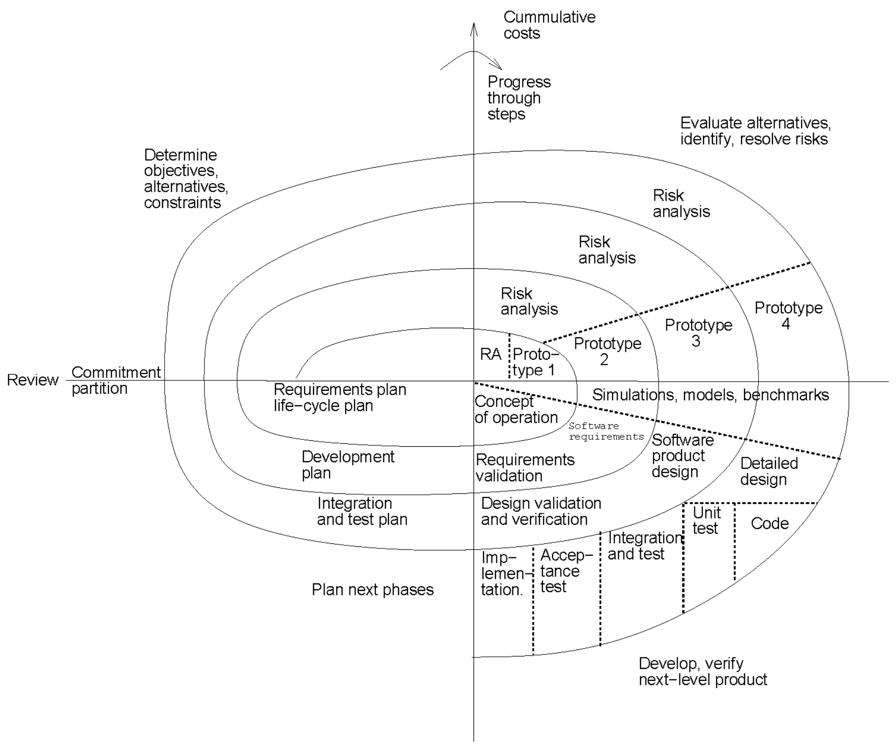


Fig. 2.7 The spiral model (adapted from (Boehm 1988))

As shown in Fig. 2.7 the project is intended to iterate through a number of basic steps. Each iteration encompasses some objectives to be solved, and is comprised of the following steps:

- Determine objectives, alternatives, and constraints.
- Evaluate the alternatives and identify risks connected to central components or features.
- Develop product to resolve the most critical risks, and evaluate the results. Prototyping, reuse techniques, and requirement and design specifications are all means of reducing risks.
- Plan the next iteration and review the achievements of the current one.

According to Boehm the strengths of the spiral model are:

- It focuses early attention on options involving the reuse of existing software.
- It accommodates for life-cycle evolution, growth, and changes of the software product.
- It provides a mechanism for incorporating software quality objectives into software product development.
- It focuses on eliminating errors and unattractive alternatives early.
- For each of the sources of project activity and resource expenditure, it answers the key question: How much is enough?
- It does not involve separate techniques for software development and maintenance.
- It provides a viable framework for integrated hardware-software systems development.

Its major weakness is connected to the availability of proper risk analysis techniques. As long as risk determination is more an art than an engineering discipline, the model will give a rather theoretical and superficial impression. Moreover, the iterated reviewing of current objectives may impose some troubles to the specification of contracts between customers and developers. As presented in (Boehm 1988) it is merely a framework for development and maintenance that will be difficult to apply directly by inexperienced developers.

2.2.5 Object-oriented Systems Development and the Rational Unified Process

As object-oriented development (including object-oriented analysis and design) became increasingly popular throughout the 90ties, also object-oriented development methods became more popular, especially as UML emerged as a standard for modelling of object-oriented systems. The Unified Process and RUP are examples of methodological frameworks for systems development that arose in this period. These methods exist in several variants. The activity view of the development process is exemplified in Fig. 2.8 in the form of a 'whale-diagram'. This view shows the main activities performed during a system development process.

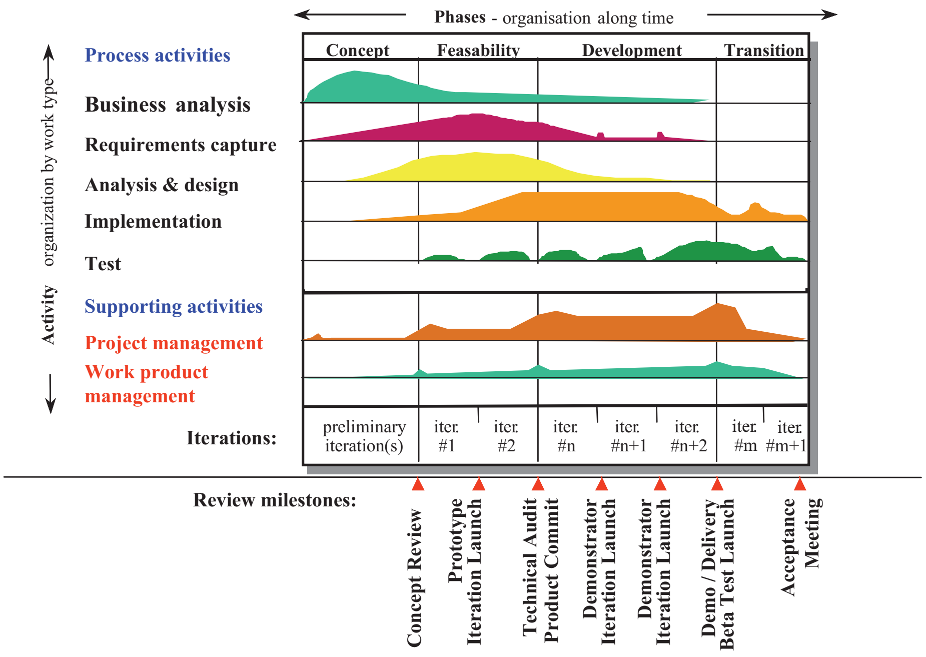


Fig. 2.8 The activity view of a development process following RUP

There are four phases in the development process: the inception phase, the elaboration phase, the construction phase and the transition phase. The completion of a phase means that the product under development has reached a certain degree of completeness and thus represents a major milestone of the project. The milestones are shown at the bottom of Fig 2.8.

There are two kinds of activities: *Process activities* and *supporting activities*. Process activities are activities that are directly related to the system development tasks. They are requirements capture, analysis & design, implementation and test. Supporting activities are activities that support the process activities to ensure that they are carried out effectively and efficiently. They are project management and work product management.

The relative importance of the process activities varies during the life cycle of a development project. In early phases, analysis and architecture level design tend to dominate, while in later phases the majority of the work is on class level design, implementation and testing. This is illustrated in Fig. 2.8 where the curves indicate the effort on each activity as a function of time.

Note that the figure shows an example. Exactly how these curves will look depends on the type of project. In more explorative projects for instance, where the requirements and architecture is difficult to stabilise early, significant effort on requirements analysis and architectural design may persist all the way to the end of the project. In more straightforward projects, on the other hand, these activities will be more or less completed after the first two phases.

The Rational Unified Process captures many of software development's best practices in a form suitable for a wide range of projects and organisations:

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Model software with visual languages
- Continuously verify software quality
- Control changes to software

2.2.6 Incremental Development and Agile Development

Incremental development (Davis 1988b) is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance. When the increments are released to the production environment one by one, one often talks about stage-wise development. This is meant to reduce the costs incurred before an initial capability is achieved. It also produces an operational system more quickly, and it thus reduces the possibility that the user needs will change during development. It also enables rapid feedback from the use of the systems to inform the work in later increments. Originally Incremental development presupposed that most of the requirements were understood up front, and one chose to implement only a subset of the requirements at a time. Note how this differs from evolutionary prototyping, even if these techniques could be integrated.

Modern approaches to incremental development practices are labelled "agile," or lightweight methodologies (Abrahamsson 2010, Cockburn 2002). Agile methodologies are based on the assumption that communication is necessarily imperfect (Cockburn 2002), and that software development is a social, communication-intensive activity among multiple developers and users of the system. According to proponents of agile methods, increased documentation is not necessarily the answer to the weaknesses of evolutionary development practices. Rather, certain complementary activities must be in place to augment evolutionary development and to increase the quality or scope of iterations, such as pair programming, time-boxing, test-first development (Beck 2002).

The agile alliance defined the Agile Manifesto in 2001. The four agile values are specified as follows (Beck et al. 2001):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to Change over following a plan

Note that this do not mean that you are not doing any documentation at all, just that the focus is rather on working software.

A number of agile methods and approaches have been developed including

- Extreme Programming (XP) (Beck 1999)
- Scrum (Schwaber and Beedle 2002)
- Dynamic Systems Development Method - DSDM (Stapleton 1997)
- Feature-Driven Development - FDD (Palmer and Felsing 2002)
- Adaptive Software Development - ASD (Highsmith 2000)
- Agile Modelling - AM (Ambler and Jeffries 2002)
- Crystal (Cockburn 2002)
- Internet Speed Development -ISD (Baskerville et al 2001)
- Pragmatic programming - PP (Hunt and Thomas 2000)
- Test Driven Development - TDD (Janzen and Saiedian 2005)

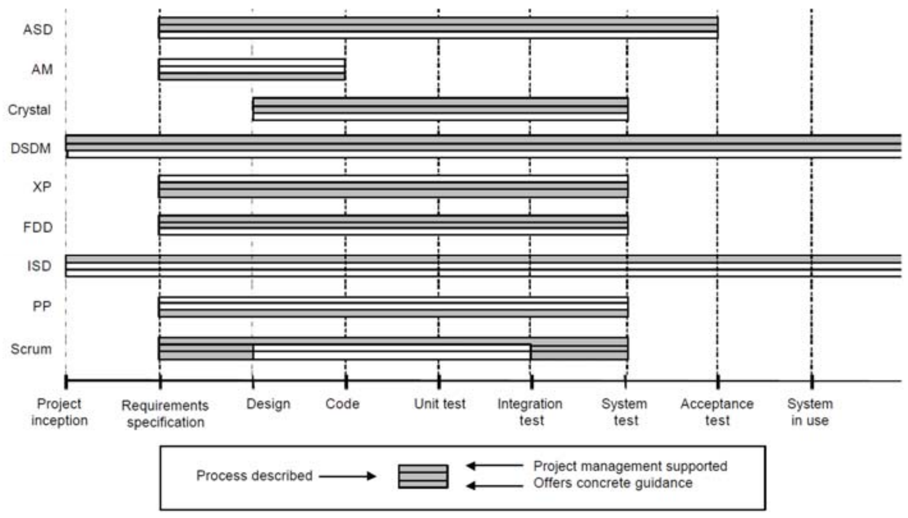


Fig. 2.9 Comparing project management, life-cycle and guidance support in agile development methods (From (Abrahamsson et al. 2010))

Often one thinks about agile methods primarily looking on coding and testing. As Fig. 2.9 from (Abrahamsson et al. 2010) illustrates, the scope and coverage of the methods differ quite a bit, although we will not go into detail on this here. One thing worth to mention though is that even if focus is being close to the end-user (or rather customer), few agile development methods relates directly to systems in use.

The below table based on discussions in (Boehm 2002, Nerur et al 2005) indicate some differences between more traditional development and agile development.

Table 2.4 Comparing agile and plan-driven development

Aspect	Agile	Traditional/Plan-driven
Developers	Agile, Knowledgeable, Collocated, and Collaborative	Plan-oriented, Adequate skills, Access to external Knowledge
Customers	Dedicated, Knowledgeable, Collocated, Collaborative, Representative, and Empowered Customers	Access to Knowledgeable, Collaborative, Representative, and Empowered Customers
Requirements	Largely Emergent, Rapid Change	Knowable early, Largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Size	Smaller Teams and Products	Larger Teams and Products
Primary objective	Rapid Value	High Assurance

One problem with strict incremental design, however, is the lack of "iterative" planning for each increment. Starting with a poor initial increment could turn users away; the focus on the main increment can contribute to a short-term, myopic focus for the project; and "developing a suboptimal system" could necessitate a great deal of rework in later phases (Boehm 1981). The output of incremental development might often resembles unmanageable "spaghetti code" that is difficult to maintain and integrate, similar to the "code and fix" problems that Waterfall was originally intended to correct (Boehm 1988) Also, by using the software code itself to guide discussions of requirements, conversations tend to focus mainly on detail, rather than business principles associated with the information system. Many continuing problems associated with incremental development include "ad hoc requirements management; ambiguous and imprecise communication; brittle architectures; overwhelming complexity; undetected inconsistencies in requirements, designs, and implementation; insufficient testing; subjective assessment of project status; failure to attack risk; uncontrolled change propagation; insufficient automation" (Kruchten 2000).

To address this, several approaches has looked upon dividing the methodology into two, one phase where the long-term architecture is developed, and one with many iterations where functionality is developed (so-called RAAD - Rapid Architected Application Development). Both in incremental/agile approaches and RUP, a focus is on bundling requirement into iterations/increments. As a basis for this, one often looks upon the priority of the requirements. Another way of structuring requirements is according to the Kano-model (Kano 1984). According to this, software requirements can be classified into three categories: Normal, exciting and expected.

Whereas many of the traditional methodologies are oriented primarily towards the development of the IT-system, a number of methodologies take a broader view, looking upon the development of the overall organisation. An early example of this was Multiview.

2.2.7 Multiview

This methodology was based on several existing methodologies (Avison and Wood-Harper 1990) Multiview addresses the following areas:

1. How is the application system supposed to further the aims of the organisation using it?

2. How can it be fitted into the working lives of the stakeholders in the organisation?
3. How can the stakeholders best relate to the application system in terms of operating it and using the output from it?
4. What information processing functions are the application system to perform?
5. What is the technical specification of an application system that will come close enough to addressing the above questions?

Multiview largely addresses problems associated with the analysis and design activities in application systems development. It tries to take into account the different views of the stakeholders.

The methodology sees information systems as social system that relies to an increasing extent on computer technology. Multiview includes phases for addressing both human/social and technical aspects.

The methodology is based on five main phases (Figure 2.10):

1. Analysis of human activity (answer to question 1).
2. Analysis of information (answer to question 4).
3. Analysis and design of socio-technical aspects (answer to question 2).
4. Design of the human-computer interface (answer to question 3).
5. Design of technical aspects (answer to question 5).

Not all development projects go through the same phases since the surroundings and particular circumstances differs from case to case. Multiview forms a flexible framework since different tools are available and are adjusted to different situations.

Analysis of human activity is based on the work of Checkland (Checkland 1981, Checkland and Scholes 1990) on Soft Systems Methodology (SSM). It focuses on finding the different stakeholders view of the world.

Developers will with help from the users form a *rich picture* of the problem situation. Based on the rich picture the problems to be investigated in more detail may be extracted. The developers imagine and name systems that might help revealing the cause of the problem. Among suggested systems the developers and the users have to decide on a relevant system that is appropriate for the actual situation. Rich pictures are based on root definitions. A root definition is a concise verbal description of the system, which captures its essential nature. One technique to help come up with the root definitions is the 'CATWOE'-technique that answers the following question:

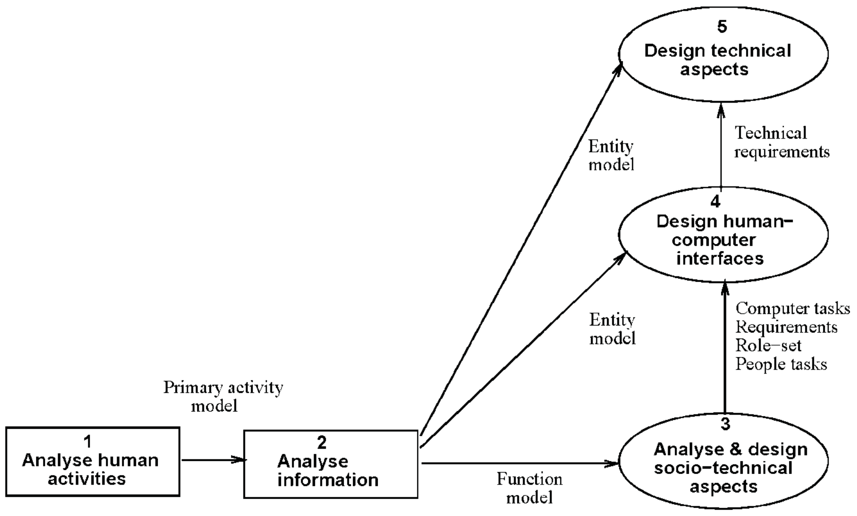


Fig. 2.10 The phases and interdependencies in Multiview

Who is doing *what* for *whom*, and to whom are they *answerable*, what *assumptions* are being made, and in what *environment* is this happening?

- *C*ostumer is the 'whom',
- *A*ctor is 'who',
- *T*ransformation is 'what',
- *W*eltanschauung is 'assumptions',
- *O*wner is the 'answerable',
- *E*nvironment is the environment.

1. An activity model show how the various activities are related to each other temporally. The activity model is a semi-formal conceptual model with some similarities to a DFD.
2. In information analysis, the three main tasks are development of a functional model, a data model, and interacting functions and entities and verifying the models.
3. Analysis and design of socio-technical aspects is concerned with the people using the IS. In order to develop a successful system, the system must be fitted into the working live of the users. The socio-technical methodology means that the technical and social aspects must fit each other in order to construct the best system. This phase is based on ETHICS (Mumford 1983)

4. Design of human-computer interfaces is concerned with the way users communicate with the CIS. Prototyping of user interfaces described above) is supported.
5. In the design of technical aspects a technical solution is created in accordance with the requirements specified in early phases. What is considered is the entity model (phase 2), computer tasks (phase 3) and the human computer interfaces (phase 4).

2.2.8 Methodologies for Maintenance and Evolution of IS

Whereas most methodologies focus on development of new systems, some also focus on maintenance and application management. An early example was the CONFORM - method. CONFORM (Configuration Management Formalisation for Maintenance) (Capretz and Munro 1994) is a method that provides guidelines for carrying out a variety of activities performed during maintenance. The method accommodates a change control framework around which software configuration management (SCM) is applied. The aim is to exert control over an existing application system while simultaneously incrementally re-documenting it. In order to enforce software quality, a change control framework has been established within CONFORM called the software maintenance model (SMM).

Below is an overview of the individual phases of SMM

- Change request: If the proposed change is corrective, a description of the error situation is included. For other types, a requirements specification is submitted.
- Change evaluation: Whereas a rejected proposed change is abandoned, a change approval form is created for an approved change. This together with the corresponding change proposal is the basic tool of the change management. By documenting new requirements, these forms become the contract between the requester of the change and the maintainers. The approved changes are ranked and selected for the next release. Changes are batched by system releases. The work required is classified as perfective, adaptive, corrective, or preventive. The inadequacies described in the change proposal are identified in the application system.
- Maintenance design phase: The result of this phase is the maintenance specification form. The design of a modification requires an examination of side-effects of changes. The maintainer must consider the software components affected and ensure that component properties are

kept consistent. The integration and system test need to be planned and updated. Additionally, if the changes require the development of new functionality, these are specified.

- Maintenance design re-documentation: This phase, along with the next, facilitates system comprehension by incremental re-documentation. The forms associated with these phases aim at documenting the software components of an existing application system.
- Maintenance implementation.
- System release: Validation of the overall system is achieved by performing integration and system test. The configuration release form contains details of the new application.

Basili (1990) illustrates how development and maintenance can be merged in a methodology

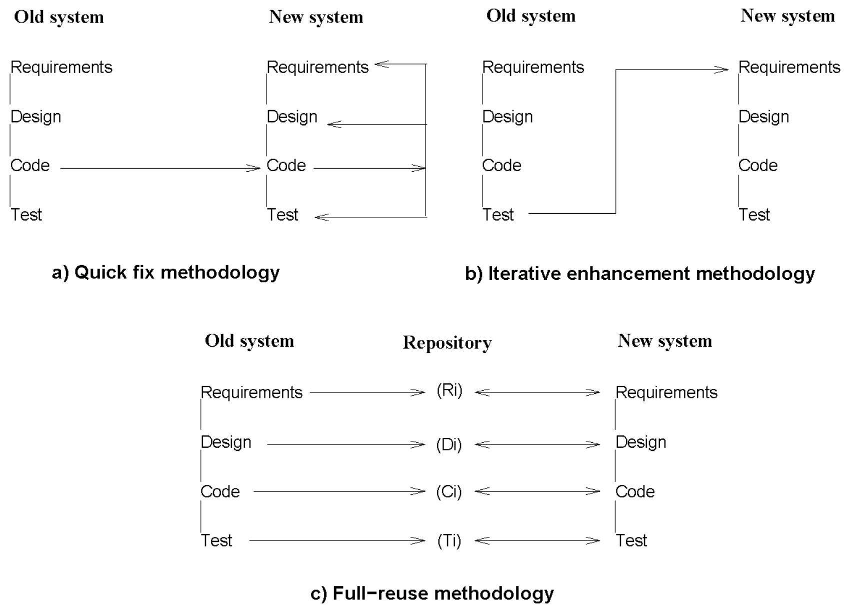


Fig. 2.11 Maintenance process models

According to Basili all maintenance is in a sense reuse. One can view a new version of an application system as a modification of the old or a new system that reuses much of the old system. Basili describes three maintenance models (Fig. 2.11).

Quick fix methodology: The existing application system, usually just the source code, is taken as outset. The code and accompanying documentation is changed, and a re-compilation gives a new version. Fig. 2.11a demonstrates the change of the old system's source code to the new version's source code.

Iterative-enhancement methodology: Although this was originally proposed for development, it is well suited for maintenance. It assumes a complete and consistent set of documents describing the application system and

- starts with the existing requirements, design, code, and test documents.
- modifies the set of documents, starting with the highest level document affected by the changes, propagating the changes down through the full set of documents.
- at each step let you redesign the application system, based on the analysis of the existing system. An environment that supports the iterative-enhancement methodology
- also supports the quick-fix methodology.

Full-reuse methodology: A full reuse methodology starts with the requirements analysis and design of the new application system and reuses the appropriate requirements, design, and code from any earlier versions of the old system. It assumes a repository of artefacts defining earlier versions of the current application system and similar systems.

An environment that supports the full-reuse methodology also supports the two other methodologies. According to Basili, one can consider development as a subset of maintenance. Maintenance environments differ from development environments in the constraints on the solution, customer demand, timeliness of response, and organisation. Traditionally, most maintenance organisations were set up for the quick-fix methodology, but not for the iterative enhancement or full reuse methodology, since they are responding to timeliness. This is best used when there is little chance the system will be modified again. Its weaknesses are that the modification is usually a patch that is not well-documented, the structure of the application system has partly been destroyed, making future evolution of the system difficult and error-ridden, and it is not compatible with development. The iterative-enhancement methodology allows redesign that lets the application system evolve, making future modifications easier. It is compatible with traditional development methodologies. It is a good methodology to use when the product will have a long life and evolve over time. In this case, if timeliness is also a constraint, one can use the quick-fix technique for patches and the iterative-enhancement methodology for long-term change. The drawbacks are that it is a more costly and possibly less timely technique in the short run, and provides little support for generic artefacts or future similar systems.

The full-reuse methodology gives the maintainer the greatest degree of freedom for change, focusing on long range development for a set of application systems, which has the side effect of creating reusable artefacts of all kinds for future developments. It is compatible with development, and is according to Basili the way methodology should evolve. It is best used when you have multi-product environments or generic development where the portfolio has a long life. Its drawback is that it is more costly in the short run and is not appropriate for small modifications, although you can combine it with other models for such changes.

In practice, large organisations have a number of applications that will follow different approaches according to the stage the application is in the life-cycle (development, evolution, servicing, phase-out, closed (Rajlich and Bennett 2000).

Over the last years, standard approaches such as Information Technology Infrastructure Library (ITIL) (Hochstein et al. 2005) has become popular supporting not only application management, but also service management more generally when providing IT-services. ITIL is a framework of best practices to manage IT operations and services. Government of Commerce, UK defined ITIL in the mid 1980s . ITIL 's main objective is to align business and Information Technology. ITIL's IT Service Support process helps organisations to manage software, hardware, and human resource services to ensure continued and uninterrupted business. ITIL defines that the core function of IT Service is to offer "uninterrupted and best possible service" to all users. It defines 5 processes: Incident Management, Problem Management, Configuration Management, Change Management, and Release Management. ITIL does not mandate enterprises and organisation to implement all the framework specifications. This freedom to choose is one of the prime reasons why ITIL is still very relevant even today to enterprises of all sizes.

2.2.9 Enterprise Architecture

Whereas most methodologies described so far relates to development and evolution of IT-systems, other methodologies take a wider approach, looking on the whole enterprise. This is often discussed under the heading of enterprise architecture.

A number of enterprise architectures approaches exist, including:

1. The Zachman Framework from the Zachman Institute for Architecture (Sowa and Zachman 1992, Zachman 1987)
2. The GERAM Framework from The University of Brisbane (Bernus and Nemes 1996)
3. Archimate (Lankhorst 2005)
4. ARIS (Architecture of Integrated Information Systems) from IDS Scheer (Scheer 1999).
5. The CIMOSA Framework from CIMOSA GmbH (Zelm 1995).
6. The DoDAF Architecture Methodology from the FEAC Institute (DoD 2003)
7. TOGAF Architecture Methodology from the Open Group (TOGAF 2011)

We briefly describe two of these, Zachman and TOGAF. A short description of the others can be found in (Lillehagen and Krogstie, 2008).

The Zachman Framework for Enterprise Architecture

The Zachman framework as it applies to enterprises is simply a logical structure for classifying and organising the descriptive representations of an enterprise that are significant to the management of the enterprise as well as to the development of the enterprise's systems.

The framework graphic in its most simplistic form depicts the design artefacts that constitute the intersection between the roles in the design process, that is, Owner, Designer and Builder; and the product abstractions, that is, What (material) it is made of, How (process) it works and Where (geometry) the components are, relative to one another. These roles are somewhat arbitrarily labelled Planner and Sub-Contractor and are included in the Framework graphic that is commonly exhibited.

From the very inception of the framework, some other product abstractions were known to exist because it was obvious that in addition to What, How and Where, a complete description would necessarily have to include the remaining primitive interrogatives: Who, When and Why. These three additional interrogatives would be manifest as three additional columns of models that, in the case of enterprises, would depict:

- Who does what work,
- When do things happen and
- Why are various choices made.

A balance between the holistic, contextual view and the pragmatic, implementation view can be facilitated by a framework that has the characteristics of any good classification scheme, that is, it allows for abstractions intended to:

- simplify for understanding and communication, and
- clearly focus on independent variables for analytical purposes, but at the same time,
- maintain a disciplined awareness of contextual relationships that are significant to preserve the integrity of the object.

In summary, the framework is meant to be:

- Simple i.e. it is easy to understand. It is not technical, but purely logical. Anybody (technical or non-technical) can understand it.
- Comprehensive i.e. it addresses the enterprise in its entirety. Any issues can be mapped against it to understand where they fit within the context of the enterprise as a whole.
- A language - it helps you think about complex concepts and communicate them precisely with few, non-technical words.
- A planning tool - it helps you make better choices as you are never making choices in a vacuum. You can position issues in the context of the enterprise and see a total range of alternatives.
- A problem solving tool - it enables you to work with abstractions, to simplify, to isolate simple variables without losing sense of the complexity of the enterprise as a whole.
- Neutral - it is defined independently of tools or methodologies and therefore any tool or any methodology can be mapped against it to understand their implicit trade-offs, that is, what they are doing, and what they are not doing.

TOGAF Architecture Methodology

TOGAF (The Open Group Architecture Framework) has from its early days, 1997, been developed and owned by the Open Group, an international interest organisation. It now has a strong position with private industry in the US, Britain and Japan.

The present version of TOGAF being offered is version 9. TOGAF has had a good certification and training services in place since version 7. Most enterprise architecture tool vendors are members and have access to these versions, and to services helping them to qualify as authorised and certified TOGAF compliant providers of the methodology, and to get access to other services like training, consulting and events participation.

TOGAF has itself an interesting architecture. It offers an Architecture Development Methodology (ADM) as a separate model following the steps depicted in Fig.212. Popkin System Architect has the most comprehensive model of the TOGAF methodology allowing visual navigation of all core domains and their constructs and relationships to other constructs and domains, such as: strategies, proposed initiatives, present IT portfolio, present systems and their use, users and vendors, all systems, their capabilities and use, and support for searching, view-generation, reporting, and "what-if" analysis.

Most leading Enterprise Architecture vendors are supporting TOGAF, such as Troux Architect from Troux, System Architect from Popkin, and Mega.

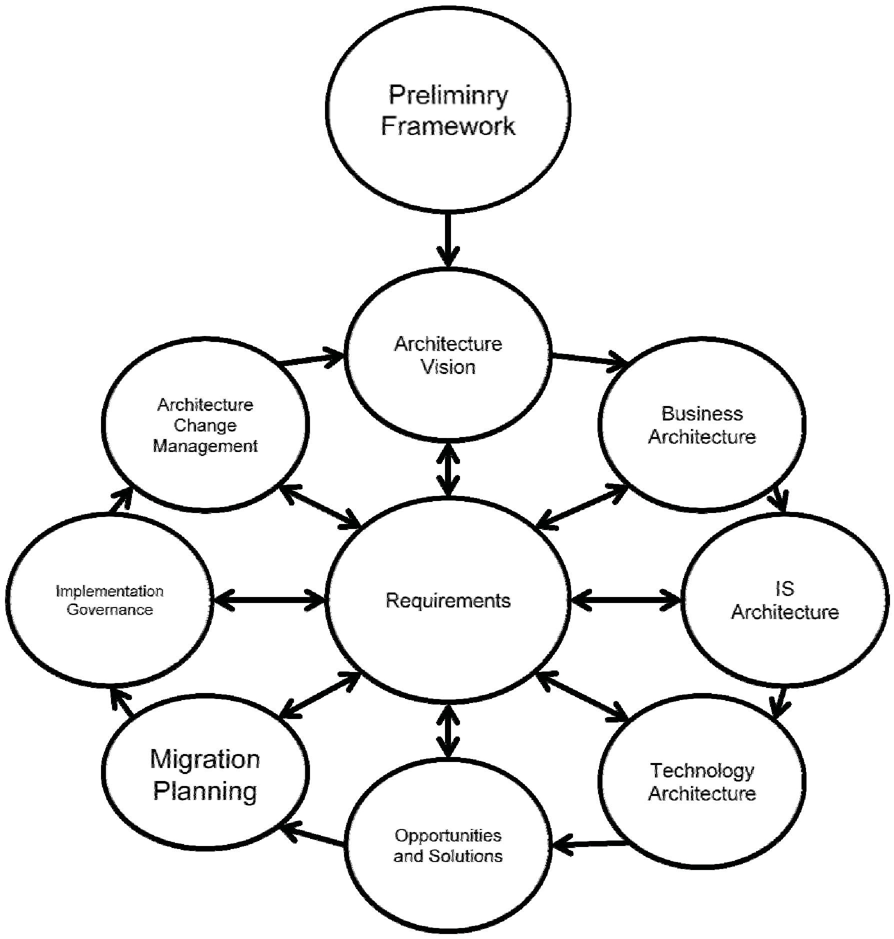


Fig. 2.12 Main steps in a TOGAF architecture development project

2.2.10 Complete Methodological Framework

With a complete framework, we refer to methods with a full coverage of product and process. As focus has shifted from developing technical application to providing value for organisations, also methodologies has gone from to encompass changes across the whole enterprise architecture to ensure the delivery and evolution of solutions in a managed way. Towards the end of the nineties consultancy companies had developed integrated frameworks that had to be adapted from project to project. As an example we provide some highlights of BIM - Business Integration Methodology.

BIM looked at the development of strategy, organisational structure, process and technology in an integrated manner, involving:

- Management
    - Change management
    - Program management
    - Project management
- Planning
    - Strategic diagnosis
    - Strategy development
    - Operating strategy
    - Enterprise Architecture
- Delivery
    - Analysis
    - Design
    - Implementation
    - Deployment
- Operations
    - Service management
    - Application management (including maintenance)

2.3 Examples of Model-based Methodologies

A number of artefacts are developed in different methodologies for developing and evolving information systems. These will differ, including informal natural language, semi-formal and formal two-dimensional (conceptual) models, and operational code. Given the theme of the book, we focus in particular on the role of conceptual models, both in development, use, and evolution.

2.3.1 Traditional use of Modelling in Analysis, Requirements Specification and Design

Modelling has traditionally been used in analysis, requirements and design as a documentation method to be used as input for further development.

In structured analysis (Yourdon 1988) one follow a traditional waterfall approach, enriching this with a set of graphical documentation/modelling techniques to specify the functional requirements in a top-down manner:

- Data flow diagrams (DFD) (Gane and Sarson 1979) document the overall functional properties of the system. Data flow diagrams are described in more detail in Sect. 3.3.3.
- Entity relationship (ER) diagrams (Chen 1976) or models in a similar semantical data modelling language model entities and the relationship between these entities. ER-diagrams are described in m ore detail in Sect 3.3.4.
- A data dictionary is used to record definitions of data elements.
- State transition diagrams (STD) may be used to specify the time-dependent behaviour (control structures) of the system. Behavioural modelling of this type is described in more detail in Sect. 3.3.2.
- Process specifications can be written in a variety of ways: decision tables, flowcharts, graphs, ``pre'' and ``post'' conditions (rules), and structured English.

Structured design is defined as ``the determination of which modules, interconnected in which way, will best solve some well-defined problem'' (Yourdon 1988). It is assumed that structured design has been preceded by structured analysis. The design process is guided by design evaluation criteria and design heuristics, resulting in a set of structure charts.

A number of text-books exist in this area (e.g. (Avison and Fitzgerald 2006, Hawryszkiewycz 2001, Marakas 2006, Lejk and Deeks 2002, Valacich et al. 2012) and we expect this material to be well known by those reading this book and do not use more space on it here. We will return to the main modelling languages from this tradition in Chap. 3.

2.3.2 MDA - Model Driven Architecture

Model-driven architecture (MDA^{tm}) has become OMG's notion of doing model-driven development, and has gained a lot of interest (Miller et al.

2003). MDA has for some time being used in some practical case studies in industry, also on the enterprise level (Günther and Steenbergen 2004).

Model-driven Architecture (MDA) can be looked upon as a variant of model-driven development (MDD) which represents an approach to system engineering where models are used in the understanding, design, construction, deployment, operation, maintenance and modification of software systems. Model transformation tools and services are used to align the different models, ensuring that they are consistent across different refinement levels, and such it resembles the transformational approach described in section 2.2.3.

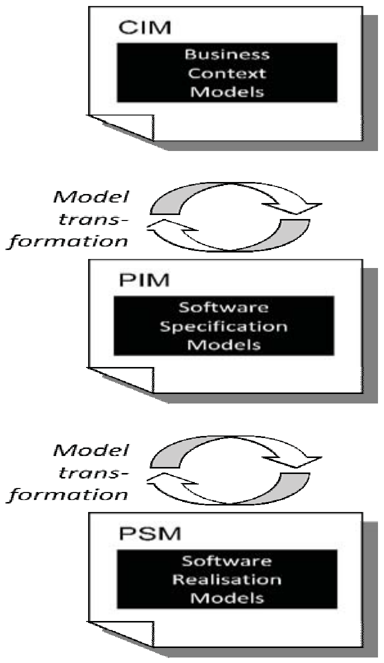


Fig. 2.13 Levels of models in MDA

Model-driven development in this view represents a business-driven approach to software systems development that starts as illustrated with a computation independent model (CIM) describing the business context and business requirements. The CIM is refined to a platform independent model (PIM) which specifies services and interfaces that the software systems must provide to the business, independent of software technology platforms. The PIM is further refined to a platform specific model (PSM) which describes the realisation of the software systems with respect to the chosen software technology platforms. In addition to the business-driven approach, a model-driven framework should also address how to integrate and modernise existing legacy systems according to new business needs. This approach is known as architecture-driven modernisation (ADM) in the OMG.

The three primary goals of MDA are portability, interoperability and reusability. The MDA starts with the well-known and long established idea of separating the specification of the operation of the system from the details of the way the system uses the capabilities of its software execution platform (e.g. J2EE, CORBA, Microsoft .NET and Web services).

MDA provides an approach for:

- specifying a system independently of the software execution platform that supports it;
- specifying software execution platforms;
- choosing a particular software execution platform for the system;
- transforming the system specification into one for a particular software execution platform;

Model-driven development is concerned with using the appropriate set of models and modelling techniques, supported by the appropriate tools, to provide sufficient help for reasoning about systems. In MDA, one takes as a outset that one is using UML for modelling, although different parts of UML is relevant at different levels. One also often needs to adapt UML, especially for the representation of PSMs. MDA defines a metamodel hierarchy for modelling a system. A system is described by a model (at the M1 level). A model conforms to a metamodel (at the M2 level) which defines the modelling constructs used in the model. The metamodel itself is described in a common meta-metamodel language (at the M3 level). The meta-metamodel language in the OMG MDA is MOF. MOF defines the core modelling constructs needed to describe all metamodels of interest to model-driven development. MOF can e.g. be used to describe the UML metamodel and adaptations of this for supporting specialisations of UML. Meta-model levels are further discussed in Sect. 3.1.3.

2.3.3 DSM and DSL

The essence of DSM (Domain Specific Modelling) and DSL (Domain Specific Languages) are to adapt the modelling language used to the domain to be modelled and the stakeholders' knowledge of this domain including the important concepts in the domain. Early tools enabling meta-modelling was developed already in the late eighties and early nineties (e.g. the Ramatic - tool). Later a number of environments for DSM/DSL has appeared, including Troux Architect (earlier METIS), MetaEdit (Kelly et al 1996, Kelly and Tolvanen 2008) , Eclipse (EMP), GME, and Microsoft's DSL Tools for Software Factories (Bézivin et al. 2005).

Generally two areas have been supported with DSM-tools:

- Enterprise modelling, primarily supporting sense-making and communication of the enterprise level
- Software development, support code-generation for new software systems

The term DSL (Domain Specific Languages) is traditionally used relative to the second use. In both cases, a main difference from more traditional use of modelling is that rather than using a standard modelling language, use one adapted to the particular domain. Since there is an additional cost of building the modelling language to be used, one find most examples of this approach in case where there are a number of similar products to be developed (e.g. the case study reported for Nokia by MetaEdit+ (Kelly and Tolvanen 2008). With the right domain, the following advantages have been reported for this type of approach (Kelly and Pohjonen 2003).

- Increased productivity (when the DSL has been developed). Productivity increases by as much as a factor of 5-10 has been reported
- Better flexibility and responsiveness to change
- Sharing domain expertise with the whole theme, reducing training time needed before being productive.

In connection to work on MetaEdit+ for instance, they device the following generic methodology for developing a DSL:

- Develop the domain concepts. Define the concepts and relationships between concepts that are important in the chosen domain. Concentrate in this stage on the semantics of the concepts, not how to represent them in the language. Wait with necessary aspects relative to make the language usable for code-generation. The meta-modelling in MetaEdit uses a language particularly developed for the task (GOPRR). Approaches for meta-modelling (language modelling) is described more generally in Sect. 3.1.3)
- Define domain rules i.e. the rules for how relationships can link concepts, rules for decomposition etc.

- Create the notation, i.e. which symbols to use to represent concepts and relationships between symbols
- Implement generators: This include both code generators and other generators e.g. for documentation

Although one in principle can start from scratch at developing a new language, one often take as an outset (a subset of) an existing language, which gives some constraints e.g. on how the notation can be developed.

When the language is developed, it can be used as part of software production. One should be aware that the practical usage of the approach often will necessitate further development of the language, which again will necessitate a formal version control and configuration management approach.

Newer versions of e.g. MetaEdit include functionality to deal with this situation, including (Tolvanen et al. 2007).

- Graphical and form-based metamodelling not needing programming of the meta-model environment to implement the DSL
- WYSIWYG symbol editor with conditional and generated elements
- Integrated, incremental metamodelling and modelling: models update automatically yet non-destructively when the metamodel changes
- Support for handling multiple integrated modelling languages
- Support for multiple simultaneous language developers (meta-modelers) when developing and maintaining the DSL
- Generator editor and debugger integrated with metamodel and models
- Straight model-to-code transformations not needing to go through intermediate formats
- Generator that can map freely between multiple models and multiple files
- Support for multiple simultaneous modelers, enabling free reuse across models
- Being able to runs on all major platforms, and integrate with any IDE
- Live relations between code and models, being able to click generated code to see the original model element

In Sect.7.4, we look upon how one can use general knowledge about quality of models and modelling language (described in chapter 4 and 5) for developing new or adapted modelling languages.

2.3.4 Business Process Modelling (BPM) and Workflow Modelling

Business Process Modelling is the activity of representing both the current ("as is") and future ("to be") processes of an enterprise, so that the current process may be analysed and improved. It focuses on business processes as a sequence/three of executions in a business context based on the purpose of creating goods and services (Scheer 1999). BPM is typically performed by business analysts and managers who are seeking to improve process efficiency and quality. The process improvements identified by BPM may or may not require IT involvement, although that is a common driver for the need to model a business process.

Business Process Modelling plays an important role in the business process management (BPM) discipline, a more wide-ranging method of aligning an organisation with the wants and needs of clients.

Modelling language standards that are used for BPM include Business Process Modelling Notation (BPMN), Business Process Execution Language (BPEL), and Web Services Choreography Description Language (WS-CDL). BPM addresses (as one of many possible approaches) the process aspects of an Enterprise Architecture (see below).

BPM solutions are built on and extend the idea of workflow. WfMC (Workflow Management Coalition a non-profit, international organisation of workflow vendors, users, analysts and university/research groups) was founded already in 1993, to develop and promote workflow integration capability. A Workflow Management System is a system that defines, manages and executes “workflows” through the execution of software whose order of execution is driven by a computer representation of the workflow logic.

Three functional areas are supported by WfMC:

- The build-time functions, concerned with defining, and possibly modelling, the workflow process and its constituent activities
- The run-time control functions concerned with managing the workflow processes in an operational environment and sequencing the various activities to be handled as part of each process (the workflow engine)
- The run-time interactions with human users and other IT applications for processing the various tasks

In BPM, different aspects of this are typically replaced by specific technologies and modelling approaches (Havey, 2005). The only part in his conceptualisation that is purely modelling-oriented is BPMN, which are described in more detail in Sect. 3.3.3.

According to (Havey, 2005), a difference between workflow and BPM is that instead of sending messages between processes based on process ID's, using e.g. BPEL a process knows to accept a message intended for it based on some aspect of the message. In addition, process flow is decentralised, not dependent on a central workflow engine. BPM is often integrated with the application of SOA - Service Oriented Architecture. SOA aims to break up monolithic applications into reusable component services that can be put together in new ways to support emerging business needs.

Many projects are concerned with the development of service-oriented solutions that can be more easily planned and then later customised when they are being deployed. This is intended to provide better industry focused solutions that can be adapted better for deployment into client environments. This is to directly counter the problem of high costs of customisation and integration of highly generic industry solutions

There are two basic approaches to BPM and process improvement: (1) Business Process Reengineering (BPR) as a radical redesign of business processes by a singular transformation (Hammer and Champy 1993) and (2) evolutionary improvement of business processes by continuous transformation. Today the latter is the most important for practical BPM efforts (Weske 2007). Models are important in all phases, since the models used in definition and modelling typically are implemented through semi-automatic or automatic transformation to the executional level. In (Weske 2007) an overall approach is provided, with the following steps:

- Strategy and organisation
- Survey: Define project goals, establish project team, gather information on the business environment
- Design : Represent information as business process models
- Platform selection
- Implementation and test
- Deployment
- Operations, monitoring and control

Results from operations often are used in connection to continuous improvement, and eventually for further strategy work. This continuous improvement approach is typically conceptualised by a BPM Life Cycle (Houy et al. 2010).

More agile approaches to process modelling exist. The limited success of traditional WfMS in supporting knowledge intensive and cooperative work, has partly been attributed to lack of flexibility (Agostini and Michelis 2000) . Most work within the area of flexible workflow looks at how conventional systems can be extended and enhanced, how static workflow systems can be made *adaptive*. Research challenges for adaptive workflow include:

- Controlled handling of exceptions
- The dynamic change problem: migration of instances from an old workflow model to a new one.
- Late modelling during enactment, and local adaptation of particular workflow instances.

Most researchers in this area recognise that change is a way of life in organisations, but a basic premise is still that work is repetitive and can be relatively completely prescribed.

Within the community, an understanding seems to have emerged that change requires process definition and process enactment to be intertwined. Still, most research on adaptive workflow is based on the premise that the enactment engine is solely responsible for interpreting the workflow model. In other words, users contribute by making alterations to the model, not by interpreting any part of the model. Thus, the model must be formally complete to prevent ambiguity and deadlock from paralysing the process. Jørgensen (2001) and Weber et al. (2009) identify a range of issues related to more dynamic provisioning of process support than usually found in BPM solutions. This involves:

- Providing good usability and pragmatic quality of models and user interfaces.
- Validation and activation of ad-hoc changes to models.
- Support for changes at a high-abstraction level and creating change patterns updating several parts of the model.
- Process schema evolution, version control and migrating process instances.
- Providing coordination support (enactment, awareness).
- Making changes made to process instances reusable.
- Resolving access control to different parts of the model, handling concurrency control for conflicting changes performed by different users, as well as resolving ambiguities and exception handling.
- Model maintenance and administration of model repositories.
- Enterprise resource management, and horizontal resource allocation.
- Supporting communication between actors in virtual organisations.
- Allowing for local variants and domain specific models at the agency-level synchronised to the service process.
- Diagnosing and mining based on process instance variants.

- Traceability of changes and monitoring of dynamic processes (transparency and trust).
- Orchestration of multi-threaded cross-agency process instances.

In interactive workflow one try to address some of these issues, and we will describe this relative to how it is supported in AKM in section 2.3.6 below.

2.3.5 Enterprise Modelling

Enterprise modelling (Fox and Grüninger 2000, Vernadat 1996) is used for externalising, making and sharing enterprise knowledge, which is again vital to support enterprise systems evolution. Enterprise modelling has been defined as the art of externalising enterprise knowledge, i.e. representing the core knowledge of the enterprise (Vernadat 1996). A crucial problem for the successful evolution of enterprise systems (and thereby of enterprises) is that management have a limited understanding of their own business processes (Dalal et al. 2004), and it is argued that this can be helped by making processes and other parts of the organisation explicit in models. The importance of EM is also evident in the increasing interest for *enterprise architecture* e.g. (Pereira and Sousa 2004), which has revitalised past research on information systems architecture (Zachman, 1987). Some examples of approaches to Enterprise Architecture where provided in section 2.2.8 above, and will not be reiterated here. Another framework, GERAM will be described in some more detail.

GERAM (Generalized Enterprise Reference Architecture and Methodology) encompasses knowledge needed for enterprise engineering / integration. GERAM is describing the components needed in enterprise engineering/enterprise integration processes, such as:

- Major enterprise engineering/enterprise integration efforts (green field installation, complete re-engineering, merger, reorganisation, formation of virtual enterprise or consortium, value chain or supply chain integration, etc.);
- Incremental changes of various sorts for continuous improvement and adaptation.

GERAM is intended to facilitate the unification of methods of several disciplines used in the change process, such as methods of industrial engineering, management science, control engineering, communication and information technology, i.e. to allow their combined use, as opposed to segregated application.

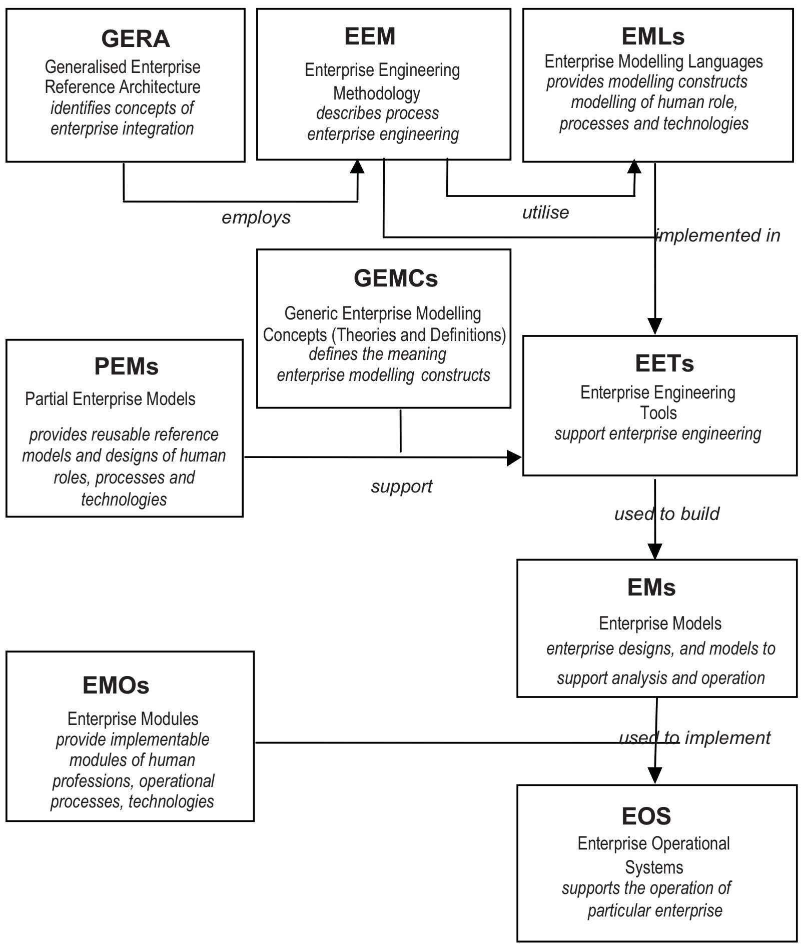

GERA
Generalised Enterprise Reference Architecture
identifies concepts of enterprise integration
EEM
Enterprise Engineering Methodology
describes process enterprise engineering
EMLs
Enterprise Modelling Languages
provides modelling constructs modelling of human role, processes and technologies
employs
utilise
implemented in
GEMCs
Generic Enterprise Modelling Concepts (Theories and Definitions)
defines the meaning enterprise modelling constructs
PEMs
Partial Enterprise Models
provides reusable reference models and designs of human roles, processes and technologies
EETs
Enterprise Engineering Tools
support enterprise engineering
support
used to build
EMs
Enterprise Models
enterprise designs, and models to support analysis and operation
EMOs
Enterprise Modules
provide implementable modules of human professions, operational processes, technologies
used to implement
EOS
Enterprise Operational Systems
supports the operation of particular enterprise

Fig. 2.14 GERAM framework components

Previous research carried out by the AMICE Consortium on CIMOSA, by the GRAI Laboratory on GRAI and GIM, and by the Purdue Consortium on PERA, (as well as similar methodologies by others) has produced reference architectures that were meant to be organising all enterprise integration knowledge and serve as a guide in enterprise integration programs. Starting from the evaluation of existing enterprise integration architectures

(CIMOSA, GRAI/GIM and PERA), the IFAC/IFIP Task Force on Architectures for Enterprise Integration has developed an overall definition of a generalised architecture (GERAM). GERAM is about those methods, models and tools, which are needed to build and maintain the integrated enterprise, be it, a part of an enterprise, a single enterprise or a network of enterprises (virtual enterprise or extended enterprise).

Fig. 2.14 depicts the main components of GERAM, which are further described below.

GERA (Generalized Enterprise Reference Architecture) defines the generic concepts recommended for use in enterprise engineering and integration projects. These concepts can be classified as:

1. Human oriented concepts: They cover human aspects such as capabilities, skills, know-how and competencies as well as roles of humans in the enterprise. The organisation related aspects have to do with decision level, responsibilities and authorities, the operational ones relate to the capabilities and qualities of humans as enterprise resource elements. In addition, the communication aspects of humans have to be recognised to cover interoperation with other humans and with technology elements when realising enterprise operations.
2. Process oriented concepts: They deal with enterprise operations (functionality and behaviour) and cover enterprise entity life-cycle and activities in various life-cycle phases; life history, enterprise entity types, enterprise modelling with integrated model representation and model views;
3. Technology oriented concepts: They deal with various infrastructures used to support processes and include for instance resource models (information technology, manufacturing technology, office automation and others), facility layout models, information system models, communication system models and logistics models.

Modelling Framework of GERA

GERA provides an analysis and modelling framework that is based on the life-cycle concept and identifies three dimensions for defining the scope and content of enterprise modelling.

1. *Life-Cycle Dimension:* providing for the controlled modelling process of enterprise entities according to the life-cycle activities.
2. *Genericity Dimension:* providing for the controlled particularisation (instantiation) process from generic and partial to particular.

3. V*iew Dimension:* providing for the controlled visualisation of specific views of the enterprise entity.

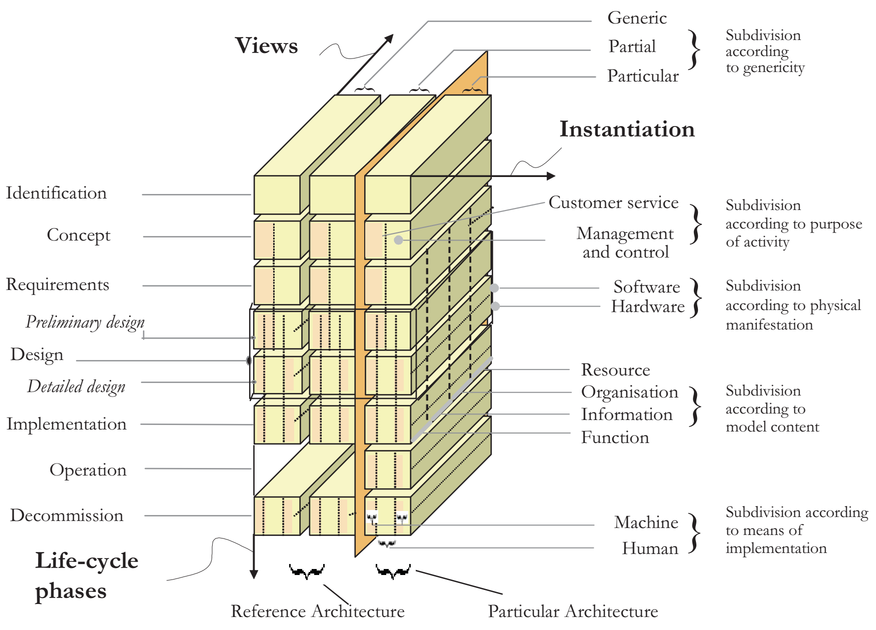


Fig.2.15 GERA Modelling Framework with Modelling Views

Fig. 2.15 shows the three dimensional structure identified above which represents the modelling framework. The reference part of the modelling framework consists of the generic and the partial levels only. These two levels organise into a structure the definitions of concepts, basic and macro level constructs (the modelling languages), defined and utilised for the description of the given area. The particular level represents the results of the modelling process - which is the model or description of the enterprise entity at the state of the modelling process corresponding to the particular set of life-cycle activities. However, it is intended that the modelling languages should support the two-way relationship between models of adjacent life-cycle phases, i.e. the derivation of models from an upper to a lower state or the abstraction of lower models to an upper state, rather than having to create different models for the different sets of life-cycle activities.

EEMs - Enterprise Engineering Methodology

Enterprise engineering methodologies describe the processes of enterprise integration. A generalised methodology like generalised architectures is applicable to any enterprise regardless of the industry involved. An EEM will help the user in the process of the enterprise engineering of integration projects whether the overall integration of a new or revitalised enterprise or in management of on-going change. It provides methods of progression for every type of life-cycle activity. The upper two sets of these activities (identification and concept) are partly management and partly engineering analysis and description (modelling) tasks.

EMLs - Enterprise Modelling Languages

Enterprise modelling language define the generic modelling constructs for enterprise modelling adapted to the needs of people creating and using enterprise models. In particular enterprise modelling languages will provide constructs to describe and model human roles, operational processes and their functional contents as well as the supporting information, tools, office and production technologies.

GEMCs - Generic Enterprise Modelling Concepts

Generic enterprise modelling concepts are the most generically used concepts and definitions of enterprise integration and modelling. Three forms of concept definition are, in increasing order of formality: Glossaries, meta-models, and ontological theories

Some requirements that must be met are as follows:

- Concepts defined in more than one form of the above must be defined in a consistent way
- Those concepts which are used in an enterprise modelling language must also have at least a definition in the meta-model form, but preferably the definition should appear in an ontological theory

PEMs - Partial Enterprise Models

Partial enterprise models (reusable reference models) are models that capture concepts common to many enterprises. PEMs will be used in enterprise modelling to increase modelling process efficiency. In the enterprise engineering process these partial models can be used as tested components for building particular enterprise models (EMs). However, in general such models still need to be adapted (completed) to the particular enterprise entity.

EETs - Enterprise Engineering Tools

Enterprise engineering tools support the processes of enterprise engineering and integration by implementing an enterprise engineering methodology and supporting modelling languages. Engineering tools should provide for analysis, design and use of enterprise models.

EMOs - Enterprise Modules

Enterprise modules are implemented building blocks or systems (products, or families of products), which can be utilised as common resources in enterprise engineering and enterprise integration. As physical entities (systems, subsystems, software, hardware, and available human resources/professions) such modules are accessible in the enterprise, or can be made easily available from the market place. In general EMOs are implementations of partial models identified in the field as the basis of commonly required products for which there is a market.

EMs – Enterprise Models

The goal of enterprise modelling is to create and continuously maintain a model of a particular enterprise entity. A model should represent the reality of the enterprise operation according to the requirements of the user and his application. This means the granularity of the model has to be adapted to the particular needs, but still allow interoperability with models of other enterprises. Enterprise models include all those descriptions, designs, and formal models of the enterprise, which are prepared in the course of the enterprise's life history.

EOSs – Enterprise Operational Systems

Enterprise operational systems support the operation of a particular enterprise. They consist of all the hardware and software needed to fulfil the enterprise objective and goals. Their contents are derived from enterprise requirements and their implementation is guided by the design models, which provide the system specifications and identify the enterprise modules used in the system implementation.

A number of other approaches to enterprise modelling have been developed over the last 10-15 years. We present briefly one here, Enterprise

Knowledge Development (EKD), indicating the scope of modelling. Other approaches will be described in more detail in later chapters. EKD builds of Tempora (described in Sect 3.3.5) and F^3 (Bubenko et al. 1994), and then further elaborated in the ELEKTRA project (22927) (Bubenko et al. 1998, Persson and Stirna 2002). EKD is an approach strongly based on involvement and participation of stakeholders (users, managers, owners …). This means that an EKD model is gradually built by its stakeholders in participatory modelling seminars, led by one or more facilitators, an approach to modelling described in section 3.4. Secondly, it is a multi-model approach involving not only a model for conceptual structures but also interlinked submodels for goals, actors, business rules, business processes, and requirements to be stated for the information system, if such is developed. All these models are interlinked, for instance a goal in a goal model may refer to a concept in the concepts model, if the concept is used in the goal description (see Fig. 2.16). The EKD approach, or other multi-model, participatory approaches similar to EKD, are now in frequent use in a wide range of practical applications and having a spectrum of purposes. We find uses of this kind of approach not only for information system development, but also for organisational development, business process analysis, knowledge management studies, and many more.

Although enterprise modelling as described here is useful both in sense-making and systems development, for modelling and model-based approaches to have a more profound effect, we propose a shift in modelling approaches and methodologies. Model-based approaches and methods must enable *regular users to be active modellers*, both when performing their work, expressing and sharing their results and values created, and when adapting and composing the services they are using to support their work. This is described below as AKM - Active Knowledge Modelling.

2.3.6 Interactive models and Active Knowledge Modelling

For the support of knowledge workers, who need more flexible support than what can be given by traditional workflow systems, it is important that emergent and interactive work processes can be captured and supported (Jørgensen 2001, Krogstie and Jørgensen 2004, Lillehagen and Krogstie 2008). The most comprehensive theoretical approach in this field is Peter Wegner's interaction framework (Wegner 1997, Wegner and Goldin 1999). The primary characteristic of an *interaction machine* is that it can pose questions to users during its computation. The process can be a multi-step conversation between the user and the machine, each being able to take the initiative.

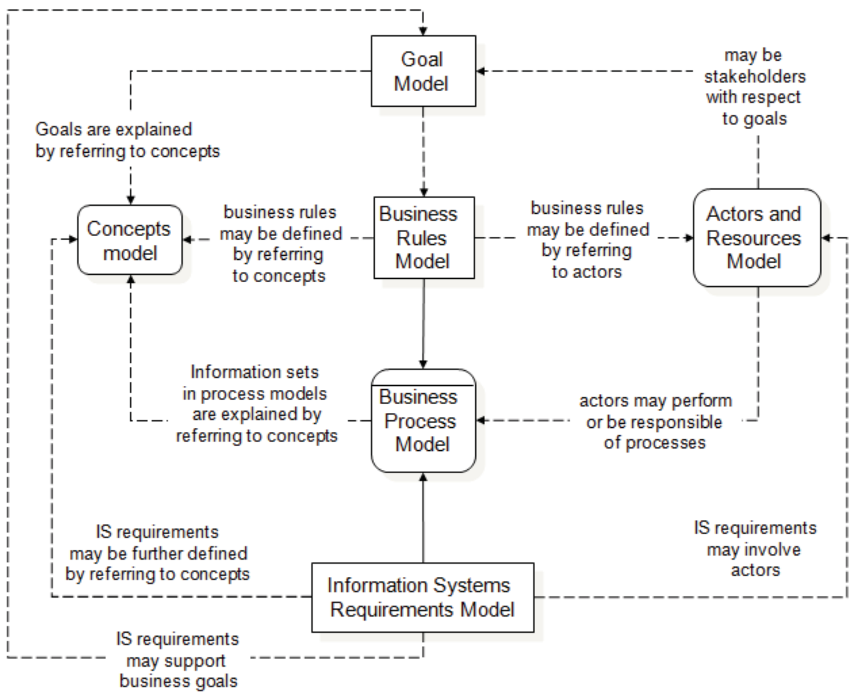

Goal Model
may be stakeholders with respect to goals
Goals are explained by referring to concepts
Concepts model
business rules may be defined by referring to concepts
Business Rules Model
business rules may be defined by referring to actors
Actors and Resources Model
Information sets in process models are explained by referring to concepts
Business Process Model
actors may perform or be responsible of processes
IS requirements may be further defined by referring to concepts
IS requirements may involve actors
Information Systems Requirements Model
IS requirements may support business goals

Fig. 2.16 A top view of the multi-model approach EKD (from (Bubenko 2007))

The Active Knowledge Modelling (AKM) (Lillehagen and Krogstie 2008) technology is about discovering, externalising, expressing, representing, sharing, exploring, configuring, activating, growing and managing enterprise knowledge. Active and work-centric knowledge has some very important intrinsic properties found in mental models of the human mind, such as reflective views, repetitive flows, recursive tasks and replicable knowledge architecture elements. One approach to benefit from these intrinsic properties is by enabling users to perform enterprise modelling using the AKM platform services to model methods, and execute work using role-specific, model-generated and configured workplaces (MGWP). Visual knowledge modelling must become as easy for designers and engineers as scribbling in order for them to express their knowledge while performing work, learning and excelling in their roles. This will also enable users to capture contextual dependencies between roles, tasks, information elements and the views required for performing work.

To be active, a visual model using an appropriate representation must be available to the users of the underlying information system at execution time. Second, the model must influence the behaviour of the computerised work support system. Third, the model must be dynamically extended and adapted, users must be supported in changing the model to fit their local needs, enabling tailoring of the system's behaviour. Industrial users should therefore be able to manipulate and use active knowledge models as part of their day-to-day work (Jørgensen, 2001, Krogstie 2007).

Recent platform developments support integrated modelling and execution in one common platform, enabling what in cognitive psychology is denoted as "closing the learning cycle". The AKM approach has at its core a customer delivery process with seven steps (C3S3P - see below). The first time an enterprise applies AKM technology, it is recommended that these steps are closely followed in the sequence indicated to establish a platform for evolution. However, second and third time around work processes and tasks from the last five steps can be reiterated and executed in any order necessary to achieve the desired results.

The AKM approach is also about mutual learning, discovering, externalising and sharing new knowledge with partners and colleagues. Tacit knowledge of the type that actually can be externalised is most vividly externalised by letting people that contribute to the same end product work together, all the time exchanging, capturing and synthesising their views, methods, properties, parameters and values, and validating their solutions. Common views of critical resources and performance parameters provide a sense of holism and are important instruments in achieving consensus in working towards common goals. The seven steps of the approach are shown in Fig. 2.17. The steps are denoted C3S3P and have briefly been described in (Stirna et al. 2007).

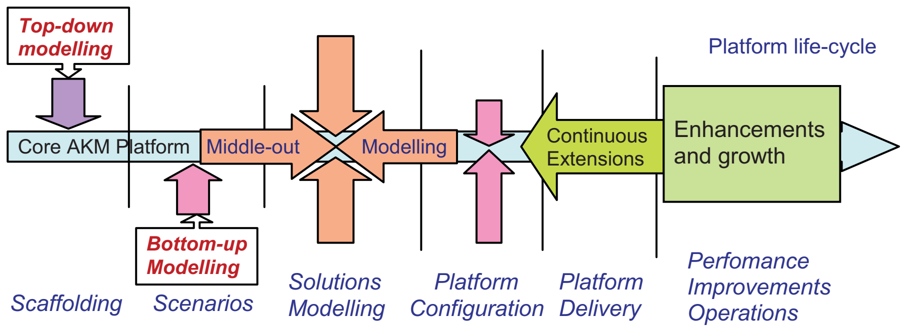


Fig. 2.17 The steps of the customer delivery process (From (Lillehagen and Krogstie 2008))

Concept testing (the C in C3S3P), performing a proof-of-concept at the customer site, is not included in the figure. The solutions modelling stage is vital for creating holistic, multiple role-views supporting work across multi-dimensional knowledge spaces, which in turn yield high-quality solution models.

Description of steps in the C3S3P methodology

1. *C*oncept testing is about creating customer interest and motivation for applying the AKM technology. This is done by running pilots and by assessing value propositions and benefits from applying the AKM approach.
2. *S*caffolding is about expressing stakeholder information structures and views, and relating them to roles, activities and systems to provide a model to raise the customer's understanding for modelling and inspire motivation and belief in the benefits and values of the AKM approach.
3. *S*cenario modelling is about modelling ”best-practice” work processes. The focus is on capturing the steps and routines that are or should be adhered to when performing the work they describe. This is the core competence of the enterprise, and capturing these work-processes is vital to perform work, support execution and perform several kinds of analyses in the solutions modelling step.
4. *S*olutions' modelling is about cross-disciplinary and cross-functional teams working together to pro-actively learn and improve quality in most enterprise life-cycle aspects. The purpose is creating a coherent and consistent holistic model or rather structures of models and sub-models meeting a well-articulated purpose. Solutions' modelling involves top-down, bottom-up, and middle-out multi-dimensional modelling for reflective behaviour and execution.
5. *P*latform configuration is about integrating other systems and tools by modelling other systems data models and other aspects often found as e.g. UML models. These are created as integral sub-models of the customised AKM platform, and their functionality will complement the CPPD methodology (see below) with PLM (Product Lifecycle Management) system functions, linking the required web-services with available software components
6. *P*latform delivery and practicing adapts services to continuous growth and change by providing services to keep consistency and compliance across platforms and networks as the user community and project networking expands, involving dynamic deployment of model-designed and configured workplace solutions and services

7. Performance improvement and operations is continuously performing adaptations, or providing services to semi-automatically re-iterate structures and solution models, adjusting platform models and re-generating model-configured and -generated workplaces and services, and tuning solutions to produce the desired effects and results.

Collaborative Product and Process Design (CPPD) as mentioned above is anchored in pragmatic product logic, open data definitions and practical work processes, capturing local innovations and packaging them for repetition and reuse. Actually most of the components of the AKM platform, such as the Configurable Product Components – CPC and the Configurable Visual Workplaces – CVW, are based on documented industrial methodologies. CPPD mostly re-implements them, applying the principles, concepts and services of the AKM platform. CVW in particular is the workplace for configuring workplaces. This includes functionality to

- Define the design methodology tasks and processes
- Define the roles participating in the design methodology
- Define the product information structures
- Define the views on product information structures needed for each task
- Perform the work in role-specific workplaces
- Extend, adapt and customise when needed

Industrial users need freedom to develop and adapt their own methodologies, knowledge structures and architectures, and to manage their own workplaces, services and the meaning and use of data. The AKM approach and the CPPD methodology support these capabilities, enabling collaborative product and process design and concurrent engineering.

2.4 Participatory Modelling

An important aspect of modelling, is to be able to represent the knowledge as held by people as directly as possible. A practical limitation earlier was that the techniques and tools used were hard to use, thus often necessitating by design or by chance the involvement of an intermediary analyst. Newer approaches have shown the possibility of involving stakeholders more directly, often with the guidance of modelling facilitators.

One approach to involvement of users is modelling conferences (Gjersvik et al. 2004). A number of other approaches exist, e.g. the use of DSL and AKM as described above and EKD and later use as described in (Persson and Stirna 2010).

2.4.1 The Modelling Conference Technique

The Modelling Conference is a method for participatory construction and development of enterprise models. In this, it takes as a starting point business process modelling to understanding how organisations work. However, while most approaches to the mapping and "re-engineering" of business processes tend to be expert and management focused, the Modelling Conference technique focuses on participation from all stakeholder groups, and the link between organisational learning and process institutionalisation through the use of technology.

The core of the Modelling Conference method has been adopted from the Search Conference method (Emery and Purser 1996). The Search Conference is a method for participatory, strategic planning in turbulent and uncertain environments. It has been used in various setting, including community development, organisation development, and the creation of research initiatives. It has also been done with a number of different designs. The method is based on a few basic ideas:

- Open systems thinking
- Active adaptation
- Genuine participation
- Learning

The concrete result of a Search Conference is a set of action plans, addressing various challenges that the conference have prioritised, and which people at the conference have committed themselves to implement. The plans may not always be congruent or coordinated, but there is a shared understanding among the participants on why each of the plans is important for parts of the system. This may be summed up in two core points:

- Action plans: "(…) multiple action plans focused on different parallel initiatives stand a better chance of diffusion than those that concentrate all their resources on one big hit." (ibid., p.63)

- Shared frame of reference: "(…) the Search Conference does not just result in more information and data about the environment. Rather, the Search Conference process also yields a shared view of the environment as conflicts or perceptual disagreements are made rational, data and information are integrated, and common ground is discovered." (ibid., p.67)

The Modelling Conference combines process modelling and search conferences, by doing process modelling in a structured conference setting, promoting broad participation. The argument for participation is discussed under Sect. 2.1.5.

A set of principles guides the Modelling Conference. The core of these principles is the ones listed for the Search Conference above, but a few are added due to the special purpose and techniques of the Modelling Conference:

- *Open systems thinking:* The unit of development (organisation, community, enterprise) is viewed as an open system, interacting with its environment. At the Modelling Conference, both the whole system itself and the main parts of the environment should be modelled. In the Modelling Conference, this might be called "open process thinking". The process is always in a context, interlinked with other processes and the rest of the contextual environment.
- *Active adaptation:* A further consequence of the open systems view is that the system needs to adapt to the environment. However, in a turbulent environment, passive adaptation is not enough. The organisation needs to influence and interact with its environment, to actively create a context in which it can develop. The participants are encouraged to think about both how they might develop the process to adapt to the demands of the customers and the context, but also what demands they might want to put on other processes and actors, including customers.
- *Genuine participation:* As in a search conference, the Modelling Conference is based on the belief that all who are part of the system or process are experts on how the system/process works as seen from their point of view. All local realities are both valid and important in constructing the common model. Given a suitable structure, the participants are themselves jointly able to analyse and understand the situation, and create suitable action plans.
- *Simplicity:* Modelling languages, methods and concepts should be simple so that it is possible for actors with various local realities to ex-

press themselves, and thus make real participation possible (Gjersvik & Hepsø, 1998)

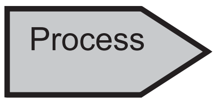
- *Pragmatism:* An important issue in the design of the conference is to find a structure and a mix of methods that will work for all participants, and which is useful in order to produce a satisfactory outcome for the actors in the organisation (Greenwood & Levin, 1998).
- *The use of the process model as a communicative and reflective device:* The models are, in addition to being the product of the conference, the main device driving the conference process. The use of large physical process visualisations encourages dialogue among the participants within a common frame of reference (Gjersvik & Hepsø, 1998).
- *Learning:* The conference should create conditions under which the participants can learn from each other, but also from the way, they work at the conference. We emphasise that the learning should not only be about the process model, but also about how to lead a discussion about the process, and about what constitutes knowledge and truth about the process and the organisation. We have used the ideas of triple loop learning (Flood and Romm, 1996), stressing that the conference is only one event in a continuous, multi-level learning process.

A Modelling Conference is performed according to the following guidelines:

- The whole process is performed in one room (or a set closely positioned rooms when parallel group-work is performed). All relevant actors in the process should be present or represented during the modelling tasks. In many cases, this also includes outside actors, like users, owners, customers, and public authorities.
- The tasks alternate between group work and plenary work.
- The participants primarily represent themselves, but are jointly responsible for the content and result of the conference.
- The staff facilitates the work, and is responsible for the method used during the conference (but not the result).
- The modelling language, tools and the overall method must be simple, so that the participants may focus on the content.
- The main outcome of the conference is a process model, which names the key processes, products and roles. Additional results are related to this process model.

The modelling language used has the following concepts and notation:

Process: A series of tasks that produce a specific product. An example is "Draw technical installations".



Product: The result of a process, and in demand by a customer. An example is the product "Drawings of technical installations", which is a product of the process mentioned above. A process may have several products; the process above may for instance also have "Documentation" as a product. We distinguish between *end products (open circle)* and *intermediate products (filled circle)*.

Customer: Someone who demands and uses the product of a process. Often, the customer is another process. For instance, the process "Install technical applications" is a customer of the process "Draw technical installations", and demands the product "Drawings of technical installations".

The conference preferably lasts at least one and a half days. Every group has a large sheet of paper on the wall, on which they work. All symbols are pre-cut, and can be attached to the sheet of paper. Through these simple symbols and physical way of working together one gets great flexibility and intensive learning, but they also limit the form of work. The results of the group work are presented in plenary sessions for discussion and joint construction of consolidated models.

The documentation from a Modelling Conference is a report and a process model. The most important outcome of the conference is the ownership that the participants develop through the construction process, which makes the model an important common reference for further more detailed development.

The conference agenda is designed so that the actors of the conference should develop models based on their own local reality before they enter a discussion with actors having (presumably) different local realities. One always start with homogenous groups, where people with the same background develop their process models. After this, the participants are more comfortable with the modelling language and tools, and have more self-confidence about their own point of view. This is especially important in organisations where there is a high risk of some groups of actors (i.e. management, experts) having model power over other participants through having a previously developed model available (Bråten, 1973). One subsequently mix the participants in heterogeneous groups, where the whole modelling starts over again.

The difficult part of the agenda is after the second modelling task, where the models of several groups are to be merged into one. This is done in a plenary session. The conference leader needs to be very attentive to the logic of the different groups, so that he or she is able to combine the elements from different models into one coherent whole. It is important that this plenary session is allowed to take the time it needs to obtain a consensus about the model.

This participatory technique has some commonalties with what is found within the field of Participatory Design (Schuler, 1993), but focuses as we have seem primarily on enterprise modelling, and not the design of technical information systems.

2.4.2 Tasks and Roles in Participatory Modelling:

In (Persson and Stirna 2010), the authors describe in more detail the processes, roles and needed competencies for a (participatory) enterprise modelling practitioner, which often is the basis for further projects including IS implementation:

The core processes and needed abilities defined (described in Sect 2.1.5 above) are (Persson and Stirna 2010)

- Define the modelling process:
    - ability to select an appropriate EM approach and tailor it in order to fit the situation at hand

- Define scope and objectives of the EM project.
    - ability to define a relevant problem
    - ability to define requirements on the results
    - ability to establish a modelling project
    - ability to navigate between the wishes of various stakeholders
    - ability to assess the impact of the modelling result and the modelling process

More about assessing the organisation and the problem at hand is available in, e.g. (Nilsson et al 1999, Stirna et al 2007)

- Plan project activities and resources.
    - ability to define requirements on the results
    - ability to navigate between the wishes of various stakeholders
    - ability to assess the impact of the modelling result and the modelling process

- Plan modelling sessions.
    - ability to define a relevant problem
    - ability to define requirements on the results

- Gather and analyse background information.
    - Interview modelling participants.
    - ability to interview involved domain experts
    - ability to navigate between the wishes of various stakeholders

- Prepare modelling session.
    - ability to define a relevant problem
    - ability to adjust a presentation of project results
    - ability to assess the impact of the modelling result and the modelling process

- Conduct modelling sessions.
    - ability to model
    - ability to facilitate a modelling session
    - ability to navigate between the wishes of various stakeholders

  More detailed recommendations of what to do and what not to do during a modelling session are available, for example, in (Jørgensen 2009, Sandkuhl and Lillehagen 2008, Stirna et al. 2007, Stirna and Persson 2009).

- Write meeting minutes. At this stage, the models should not be more refined because the main purpose of this activity is to send notes to the participants that might also serve as a reminder of the actions that they have agreed to be responsible for.

- Analyse and refine models.
    - ability to model
    - ability to define a relevant problem
    - ability to assess the impact of the modelling result and the modelling process

- Present the results to stakeholders.
    - ability to adjust a presentation of project results
    - ability to navigate between the wishes of various stakeholders
    - ability to assess the impact of the modelling result and the modelling process

2.5 Chapter Summary

In this chapter, we have looked at IS-methodology more generally providing a general classification according to

- Goal of the methodology
- Coverage of process (development, use, operations, maintenance)
- Coverage of product (from a single application to a set of co-operating organisations)
- Capabilities needed
- Stakeholder participation
- Organisation
- Location

The historical development of IS-methodologies from the first methods in the late sixties is described briefly, before we discuss in more detail some of the current modelling-oriented approaches to IS development and evolution including

- Model driven architecture (MDA)
- Domain specific modelling (DSM) and domain specific languages (DSL)
- Business process modelling/Management (BPM)
- Enterprise modelling (EM) and Enterprise Architecture (EA)
- Active Knowledge Modelling (AKM)

Review Questions

1. What are the eight areas in the framework for IS methodologies
2. We describe four main abstraction levels in IS development. What are these?
3. Describe the roles in Scrum
4. Describe competencies necessary in an enterprise modelling project
5. Mention some goals for stakeholder participation
6. List the levels of participation
7. What is the characteristics of cloud computing
8. What is the difference between the waterfall-model and the V-model?
9. What is the difference between iterative prototyping and throwaway prototyping
10. What is the difference between the transformational and operational life-cycle model
11. What are the three levels of models in MDA?
12. List 2 different approaches to DSM
13. What is the difference between workflow and BPM?
14. What are the main difference between a traditional enterprise model and an active model?
15. Describe the steps in CS3P3
16. Describe the modelling conference technique

Problems and Exercises

Individual Exercises

1. What is the differences and similarities between Agile Modelling and Incremental and iterative development
2. How can the work of (Gjersvik et al. 2004) and (Persson and Stirna 2010) on participatory modelling be combined?
3. Select one of the model-based approaches described in section 2.3. At what level(s) of abstraction relative to Fig. 1.4 is modelling performed on?

Pair Exercises

1. Discuss in pairs your own experiences with working in a project using a methodology. Classify the methodology according to level of participation.
2. Discuss in pairs your own experiences with working in a project using a methodology. Classify the methodology according to coverage in process and product
3. Discuss in pairs your own experiences with working in a project using a methodology. Classify the methodology relative to organisation and location.
4. Discuss in pairs your own experiences with working in a project using a methodology. Classify the methodology according to needed capabilities.

Group Exercises

1. Select a number of the methodologies described in Sect 2.2 and 2.3 and describe them relative to the 8 dimensions of methodologies presented in Sect 2.1.
2. Select one of the model-based approaches described in section 2.3. Look upon this approach relative the framework on types of modelling in Fig 1.3, deciding what kind of modelling tasks it focus on. You must check out some of the references or other external material to do this.
3. In connection to a modelling task, (e.g. some of the later modelling tasks suggested in this book), arrange a modelling conference involving relevant stakeholders.
4. Choose one of the modelling methodologies described in this chapter. Find if the methodology is described in Wikipedia. If it is, update the description based on material in this chapter and on other reputable sources. If not make an entry in Wikipedia on the topic.
5. Develop a 10-15 page state of the art paper on a modelling methodology mentioned in this chapter.

Chapter References

Aagesen G, van Veenstra AF, Janssen M, Krogstie J (2011) The Entanglement of Enterprise Architecture and IT-Governance: The cases of Norway and the Netherlands Paper presented at the HICCS 2011

Abrahamsson P, Oza N, Siponen MT (2010) Agile Software Development Methods: A Comparative Review. In: Dingsøyr T et al., (eds) Agile Software Development, Springer Berlin Heidelberg, 2010, pp. 31-59

Agostini A, Michelis GD (2000) Improving Flexibility of Workflow Management Systems, in Business Process Management, LNCS 1806, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Berlin, Germany: Springer, 2000

Agresti WW (1986) What are the new paradigms? In New Paradigms for Software Development, pages 6–10. IEEE

Ambler S, Jeffries R (2002) Agile Modelling: Effective Practices for Extreme Programming and the Unified Process: Wiley

Arnstein SR (1969) A Ladder of Citizen Participation JAIP, Vol. 35, No. 4, pp. 216-224

Avison DE, Wood-Harper AT (1990) Multiview: An Exploration in Information Systems Development. Blackwell, Oxford, England

Avison DE, Fitzgerald G (2006) Information Systems Development: Methodologies, Techniques and Tools Fourth Edition, McGraw Hill

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

Baskerville R, Levine L, Pries-Heje J, Ramesh B, Slaughter S (2001) How Internet companies negotiate quality. IEEE Computer 34(5): 51-57

Beck K (2002) Extreme Programming Explained: Embrace Change. The Agile Software Development Series, Addison-Wesley

Beck K et al. (2001) The Agile Manifesto. http://www.agileAlliance.org

Beizer B (1990) Software Testing Techniques. Second Edition, International Thomson Computer Press

Benestad HC, Anda BCD, Arisholm E (2009) Understanding software maintenance and evolution by analyzing individual changes: A literature review, Journal of Software Maintenance and Evolution: Research and Practice

Berger P, Luckmann T (1966) The Social Construction of Reality: A Treatise in the Sociology of Knowledge. Penguin

Berente N, Lyytinen K (2007) What Is Being Iterated? Reflections on Iteration in Information System Engineering Processes In: Krogstie, J., Opdahl, A., & Brinkkemper, S. (eds.). Conceptual Modelling in Information Systems Engineering: Springer Verlag

Bernus P, Nemes L (1996) A framework to define a generic enterprise reference architecture and methodology. Computer Integrated Manufacturing Systems, 9(3), 179-191

Bézivin J, Hillairet G, Jouault F, Kurtev I, Piers W (2005) Bridging MS/DSL Tools and the Eclipse Modelling Framework OOPSLA2005 International Workshop on Software Factories

Bjerknes G, Bratteteig, T (1995) User Participation and Democracy: A Discussion of Scandinavian Research on System Development Scandinavian Journal of Information Systems Vol 7 Iss 1

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

Boehm, BW (1981) Software Engineering Economics, Prentice-Hall

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

Boehm, BW (2002) Get ready for agile methods, with care. Computer, vol. 35, pp. 64-69

Bråten S (1973) Model Monopoly and Communications: Systems Theoretical Notes on Democratization. Acta sociologica, journal of the Scandinavian socialogical association, 16, 2:98-107

Bubenko jr JA, Rolland C, Loucopoulos P, DeAntonellis V (1994) Facilitating fuzzy to formal requirements modelling. In Proceedings of the First International Conference on Requirements Engineering (ICRE94), pages 154-157, Colorado Springs, USA, April 18-22 IEEE Computer Society Press

Bubenko jr JA, Brash D, Stirna J (1988) EKD User Guide., Dept. of Computer and Systems Science, KTH and Stockholm University, Sweden.: Kista. (1988)

Bubenko jr JA (2007) From Information Algebra to Enterprise Modelling and Ontologies – a Historical Perspective on Modelling for Information Systems In: Krogstie J, Opdahl A, Brinkkemper S (eds) Conceptual Modelling in Information Systems Engineering. Springer Verlag

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

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

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

Chapin N (1987) Evidence on seperately organizing for software maintenance. In Proceedings of the National Computer Conference (AFIPS), pages 517–522

Checkland PB (1981) Systems Thinking, Systems Practice. John Wiley & Sons

Checkland PB (1990) Soft Systems Methodology in Action. Wiley

Chen PP (1976) The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems, 1(1):9–36, March

Cockburn A (2002) Agile Software Development. Addison-Wesley

Codagnone C, Boccardelli P (2006) eGovernment Economics Project (eGEP): Measurement Framework, Final Version. In Egovernment Unit, D. I. S. A. M. (Ed., European Commission.

Dalal NP, Kamath M, Kolarik WJ, Sivaraman E (2004) Toward an integrated framework for modelling enterprise processes. Communications of the ACM 47(3):83-87, March

Davidsen MK, Krogstie J (2010) A longitudinal study of development and maintenance. Information and Software Technology 52:707-719

Davis AM (1988a) A comparison of techniques for the specification of external system behaviour. Communications of the ACM, 31(9):1098–1115, September

Davis AM, Bersoff EH, Comer ER (1988b) A strategy for comparing alternative software development life cycle models. IEEE Transactions on Software Engineering, 14(8):1453–1461, October

Davis AM (1995) Object-oriented requirements to object-oriented design: An easy Transition ? Journal of Systems and Software VOlume 30 Issues 1-2 July-August Pages 151-159

Dekleva SM (1992) Software Maintenance: 1990 Status. Journal of Software Maintenance Vol. 4 pp 233-247

Department of Defence (DoD) (2003) DoD architecture framework. Version 1.0. Volume I: Definitions and guidelines. Washington, DC: Office of the DoD Chief Information Officer, Department of Defense

Emery M, Purser RE (1996): The Search Conference. A Powerful Method for Planning Organizational Change and Community Action. Jossey-Bass Publishers, San Francisco, CA.

Eynon R (2007) Breaking Barriers to eGovernment: Overcoming obstacles to improving European public services. DG Information Society and Media. European Commission.

Falkenberg ED, Hesse W, Lindgreen P, Nilsson BE, Oei JLH, Rolland C, Stamper RK, Assche FJMV, Verrijn-Stuart AA, Voss K (1996) A Framework of information system concepts - The FRISCO Report, IFIP WG 8.1 Task Group FRISCO http://home.dei.polimi.it/pernici/ifip81/publications.html Cited Dec 2011

Fitzgerald B (2001): The Transformation of Open Source Software. MIS Quarterly 30(3): 587-598

Flak LS, Dertz W, Jansen A, Krogstie J, Ølnes S, Spjelkavik I (2009) What is the Value of eGovernment - and How can we actually realize it. Transforming Government: People, Process and Policy, 3(3), 220-226

Flak LS, Rose J (2005) Stakeholder Governance: Adapting Stakeholder Theory to the e-Government Field. Communications of the Association for Information Systems, 16, 642-664

Flood RL, Romm NRA (1996): Diversity Management. Triple Loop Learning. John Wiley & Sons, Chichester, UK

Fox MS, Gruninger M (2000) Enterprise modeling. AI Magazine

Følstad A, Jørgensen H, Krogstie J (2004) User Involvement in e-Government Development Projects. Paper presented at the NordiChi'2004

Gane C, Sarson T (1979) Structured Systems Analysis: Tools and Techniques. Prentice Hall

Gjersvik R, Hepsø V (1998) Using Models of Work Practice as Reflective and Communicative Devices: Two Cases from the Norwegian Offshore Industry. Paper presented at the Participatory Design Conference, Seattle, WA

Gjersvik R, Krogstie J, Følstad A (2004) Participatory Development of Enterprise Process Models. In: Krogstie J, Siau K, Halpin T (eds) Information Modelling Methods and Methodologies. Idea Group Publishers

Greenwood D, Levin M (1998) Introduction to Action Research: Social Research for Social Change. Sage

Grudin J, Pruitt J (2002) Personas, participatory design and product development: an infrastructure for engagement. Paper presented at Participatory Design Conference 2002, Malmø, Sweden. June

Günther J, Steenbergen C (2004) Application of MDA for the development of the DATOS billing and customer care system (Case study on the use of MDA for the development of a larger J2EE System). Proc. First European Workshop on Model Driven Architecture with Emphasis on Industrial Application, Univ. Twente, Netherlands, March 17-18.

Gupta A (2009) The Profile of Software Changes in Reused vs. Non-reused Industrial Software Systems Doctoral Thesis 2009:90 NTNU

Hammer M, Champy J (1993) Reengineering the Corporation: A Manifesto for Business Revolution. Harper Business, New York

Harel D (1992) Biting the silver bullet: Toward a brighter future for system development. IEEE Computer, January

Hauge Ø, Ayala C, Conradi R (2010) Adoption of Open Source Software in Software-Intensive Industry - A Systematic Literature Review, Information and Software Technology, 52(11):1133-1154

Havey M (2005) Essential Business Process Modelling, O'Reilly

Hawryszkiewycz I (2001) Introduction to Systems Analysis and Design Prentice Hall

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

Highsmith J (2000) Adaptive software development: a collaborative approach to managing complex systems: Dorset House Publishing Co., Inc

Hirschheim RA, Klein HK (1989) Four paradigms of information systems development. Communications of the ACM, 32(10):pages 1199-1216, October

Hochstein A, Zarnekow R, Brenner W (2005) ITIL as common practice reference model for IT service management: formal assessment and implications for practice Proceedings. The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service.

Houy C, Fettke P, Loos P (2010) Empirical Research in Business Process Management - Analysis of an emerging field of research. Business Process Management Journal, vol 16 no 4

Holgeid KK, Krogstie J, Sjøberg DIK (2000) A study of development and maintenance in Norway: Assessing the efficiency of information systems support using functional maintenance. Information and Software Technology 42:687-700

Iden J, Tessem B, Päivärinta T (2011) The alignment of IS development and IT-operations in system development projects: A multi---method research. Proceeding NOKOBIT 2011, Tromsø Norway

Hunt A, Thomas D (2000) The Pragmatic Programmer Addison Wesley

Irani Z, Love PED, Elliman T, Jones S, Themistocleous M (2005) Evaluating e-government: learning from the experiences of two UK local authorities. Information Systems Journal, 15, 61-82

Jansen S, Finkelstein A, Brinkkemper S (2009) A sense of community: A research agenda for software ecosystems. In: Proc. 31st International Conference on Software Engineering (ICSE), New and Emerging Research Track - Companion Volume, May 16-24, 2009, Vancouver, Canada, pp. 187-190

Janzen D, Saiedian H (2005) Test-Driven Development: Concepts, Taxonomy, and Future Direction. Computer, 2005, vol. 38, no. 9, pp. 43-50.

Jørgensen M (1994) Empirical Studies of Software Maintenance, PhD Thesis University of Oslo, Research Report 188, ISBN 82-7368-098-3.

Jørgensen HD (2001) 'Interaction as a framework for flexible workflow modelling', Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, ACM, pp. 33-41

Jørgensen HD (2009) Enterprise Modelling - What We Have Learned, and What We Have Not, in proc. of PoEM 2009, Springer LNBIP 39

Kano N (1984) Attractive quality and must-be quality, The Journal of the Japanese Society for Quality Control, April, pp. 39-48

Kelly S, Lyytinen K, Rossi M (1996) MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment. In: Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080, Springer-Verlag, pp. 1-21

Kelly S, Pohjonen R (2003) Domain-Specific Modelling for Cross-Platform Product Families Olive A et al. (eds) ER 2002 Ws, LNCS 2784 pp. 182-194

Kelly S, Tolvanen J-P (2008) Domain-Specific Modelling: Enabling Full Code Generation, John Wiley & Sons, New Jersey

Krogstie J, Sølvberg A (1994) Software maintenance in Norway: A survey investigation. Paper presented at the International Conference of Software Maintenance (ICSM`94), Victoria, Canada, September 19-23

Krogstie J, Jørgensen H (2004) Interactive models for supporting networked organisations. In: 16th Conference on advanced Information Systems Engineering. Riga, Latvia: (Springer, Berlin Heidelberg New York)

Krogstie J, Jahr A, Sjøberg DIK (2006). A longitudinal study of development and maintenance in Norway: Report from the 2003 investigation Information and Software Technology 48, 993-1005

Krogstie J (1996) Use of methods and CASE-tools in Norway: Results from a survey. Journal of Automated Software Engineering 3:347--367

Krogstie J (1995) On the distinction between functional development and functional maintenance. Journal of Software Maintenance 7:383--403

Krogstie J (2000) Process improvement as organizational development: A case study on the introduction and improvement of information system processes in a Norwegian organization. Paper presented at the NOKOBIT'2000, Bodø, Norway

Krogstie J (2007) Modelling of the People, by the People, for the People. In: Krogstie J, Opdahl A, Brinkkemper S (eds) Conceptual Modelling in Information Systems Engineering. Springer Verlag

Krogstie J (2010) Information Systems Development and Maintenance in Norway: Comparing Private And Public Sector. Paper presented at the NOKOBIT 2010

Krogstie J (2011) Business Information Systems Utilizing the Future Internet Perspectives in Business Informatics Research, 2011 - Springer

Kruchten P (2000) The Rational Unified Process: An Introduction (2nd Edition) Addison-Wesley

Kuusisto A (2008) Customer roles in business service production - implications for involving the customer in service innovation. Research report 195. Lappeenranta: Lappeenranta University of Technology

Langefors B (1963) Some Approaches to the Theory of Information Systems.. BIT, 3(34), 229 - 254

Langefors B (1967) Theoretical Analysis of Information Systems., Lund, Sweden: (Studentlitteratur

Lankhorst M (2005) Enterprise Archoitecture at Work, Springer

Lee M-GT, Jefferson TL (2005) An Empirical Study of Software Maintenance of a Web-Based Java Application, Proceedings ICSM'05

Lillehagen F, Krogstie J (2008) Active Knowledge Modelling of Enterprises, Springer

Lejk M, Deeks D (2002) An Introduction to Systems Analysis Techniques, Pearson

Lientz BP, Swanson EB (1980) Software Maintenance Management, Addison Wesley,.

Lientz BP, Swanson EB, Tompkins GE (1978). Characteristics of application software maintenance. Communications of the ACM, 21 (6) 466-471

Lonti Z, Woods M (2008) Towards Government at a Glance: Identification of Core Data and Issues related to Public Sector Efficiency. OECD Working Papers on Public Governance. OECD

Lowry MR, McCartney RD (eds) (1991) Automating Software Design, California, USA. The MIT Press

Lyytinen K (1987) A taxonomic perspective of information systems development: Theoretical constructs and recommendations. In: Boland Jr RJ, Hirschheim RA (eds) 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

Maguire M (2001) Methods to support human-centred design. International Journal of Human-Computer Studies, 55(4): p. 587-634

Marakas GM (2006) System Analysis & Design: An Active Approach. McGraw-Hill

McCracken D, Jackson M (1982) Life cycle concept considered harmful. ACM SIGSOFT Software Engineering Notes, 7(2):29–32, April

Mell P, Grance T (2009) The NIST Definition of Cloud Computing

Messerschmitt DG, Szyperski C (2003) Software Ecosystems: Understanding an Indispensable Technology and Industry, MIT Press

Miller G et al. (2003) Model driven architecture: how far have we come, how far can we go? (Panel at OOPSLA'03, Anaheim CA, October

Mohagheghi P, Conradi R (2004) An Empirical Study of Software Change: Origin, Acceptance Rate, and Functionality vs. Quality Attributes, Empirical Software Engineering, International Symposium on, pp. 7-16, 2004 International Symposium on Empirical Software Engineering (ISESE'04)

Mumford E (1983) Designing Human Systems for new technology: The ETHICS Method, Manchester Busines School

Mumford E (1986) Participation: from Aristotle to today. In: Langefors B, Verrijn-Stuart AA, Bracchi G (eds). Trends in information systems North-Holland An Anthology Of Papers From Conferences Of The Ifip Technical Committee, Vol. 8. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands 303-312

Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies Communications of the ACM, vol. 48, pp. 72-78

Nilsson AG, Tolis C, Nellborn C. (eds) (1999) Perspectives on Business Modelling: Understanding and Changing Organisations, Springer-Verlag

Nosek JT, Palvia P (1990) Software maintenance management: Changes in the last decade. Journal of Software Maintenance, 2, 157-174

Orlikowski JW, Gash DC (1994) Technological frames: Making sense of information technology in organizations. ACM Transactions on Information Systems, 12(2):174–207

Palmer S, Felsing J (2002) A Practical Guide to Feature-Driven Development (The Coad Series): (Prentice Hall PTR)

Parkhill DH (1966) Challenge of the Computer Utility. Addison-Wesley

Pereira CM, Sousa P (2004) A method to define an enterprise architecture using the Zachman framework. Proc. ACM SAC'04, Nicosia, Cyprus March 14-17 2004.

Persson A, Stirna J (2002) Why Enterprise Modelling - an Explorative Study into Current Practice. In: The 13th International Conference on Advanced Information Systems Engineering, CAiSE '02. Interlaken, Switzerland, Springer

Persson A , Stirna J (2010) Towards Defining a Competence Profile for the Enterprise Modelling Practitioner The Practice of Enterprise Modelling - Springer

Rajlich VT, Bennett KH (2000) A Staged Model for the Software Life Cycle, IEEE Computer, Vol. 33 66-71

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

Sandkuhl K, Lillehagen F (2008) The Early Phases of Enterprise Knowledge Modelling: Practices and Experiences from Scaffolding and Scoping. In: proc. of PoEM 2008, Springer LNBIP 15

Scheer AW (1999) ARIS, Business Process Framework (3rd ed.) Springer

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

Schwaber K, Beedle M (2002) Agile Software Development with SCRUM: Prentice Hall

Sousa H, Moreira H (1998) A Survey of the Software Maintenance Process. Proceedings of ICSM'98 IEEE CS Press Bethesda, Maryland, pp. 268-274

Sowa JF, Zachman, JA (1992) Extending and formalizing the framework for information systems architecture. IBM Systems Journal, 31(3)

Stapleton J (1997) Dynamic Systems Development Method: Addison Wesley

Stirna J, Persson A, Sandkuhl K (2007). Participative Enterprise Modelling: Experiences and Recommendations. In: Krogstie J, Opdahl AL, Sindre G (eds) Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings. Lecture Notes in Computer Science 4495 Springer

Stirna J, Persson A (2009) Anti-patterns as a Means of Focusing on Critical Quality Aspects in Enterprise Modelling, in proc. of BPMDS/EMMSAD 2009, Springer LNBIP 29

Swanson EB, Beath CM (1989) Maintaining Information Systems in Organizations. Wiley Series in Information Systems. John Wiley & Sons

Swartout WR, Balzer R (1982) On the inevitable intertwining of specification and implementation. Communications of the ACM, 25(7):438–440, July

Taylor T, Standish TA (1982) Initial thoughts on rapid prototyping techniques. ACM SIGSOFT Software Engineering Notes, 7(5):160–166, December

The Open Group architectural framework (TOGAF) (2011) http://www.opengroup.org/togaf/ . Cited Dec 2011

Tolvanen J-P, Pohjonen R, Kelly S (2007) Advanced Tooling for Domain Specific Modelling: Metaedit+ The 7th OOPSLA Workshop on Domain-Specific Modelling DSM'07

Valacich JS,George JF, Hoffer JA (2012) Essentials of Systems Analysis & Design Fifth Edition, Pearson

Vernadat FB (1996) Enterprise Modelling and Integration: Principles and Applications, Chapman & Hall

Vonk R(1990) Prototyping— The effective use of CASE technology. Prentice Hall.

Ward J, Daniel E (2006) Benefits Management. Delivering Value from IT Investments. Chichester: Wiley

Weber B, Sadiq S, Reichert M (2009) Beyond rigidity - dynamic process lifecycle support. Computer Science-Research and Development, vol. 23, No. 2, pp. 47-65, Springer

Wegner P (1997) Why interaction is more powerful than algorithms, Communications of the ACM, vol. 40, no. 5.

Wegner P, Goldin D (1999) Interaction as a framework for modelling, In Conceptual Modelling, ed by Chen, P.P., Akoka,J., Kangassalo, H., Thalheim, B. Lecture Notes in Computer Science 1565 Springer Berlin Heidelberg New York

Weske M (2007) Business Process Management: Concepts, Languages, Architectures. Springer, Berlin

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

Zachman JA (1987) A framework for information systems architecture. IBM Systems Journal, 26(3), 276-291

Zave P (1984) The operational versus the conventional approach to software development. Communications of the ACM, 27(2):104–117, February

Zelm M (ed) (1995) CIMOSA: A primer on key concepts, purpose, and business value (Technical Report). Stuttgart, Germany