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)

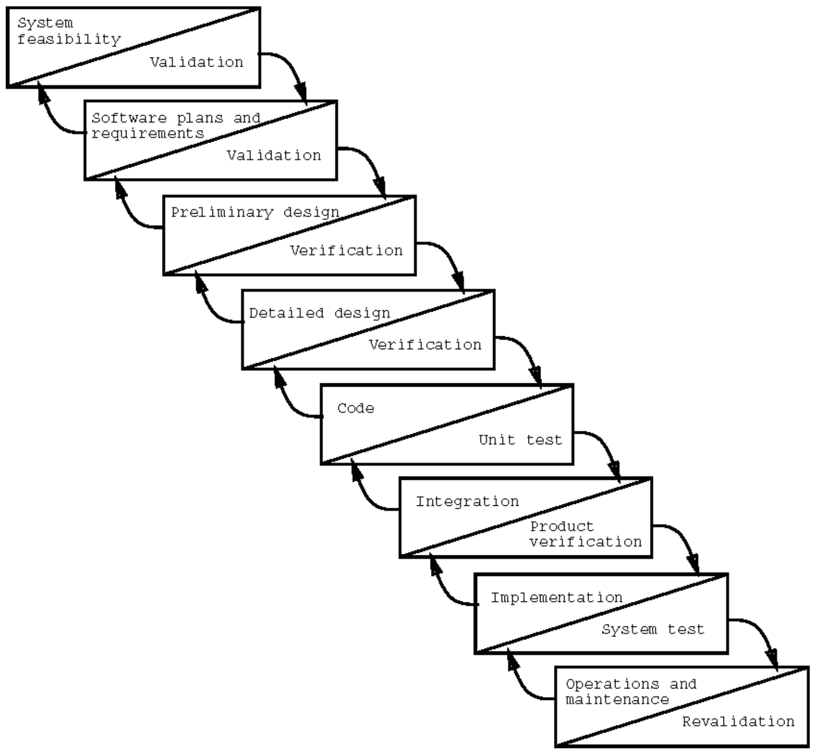

Fig. 2.3 An example of steps in a waterfall methodology

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.

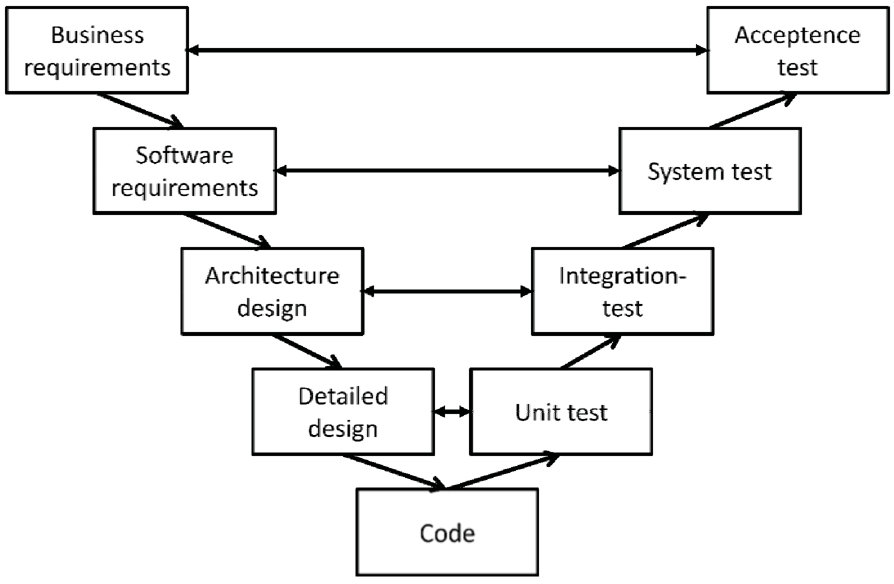

Fig. 2.4 The V-Model relating development and testing activities

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.

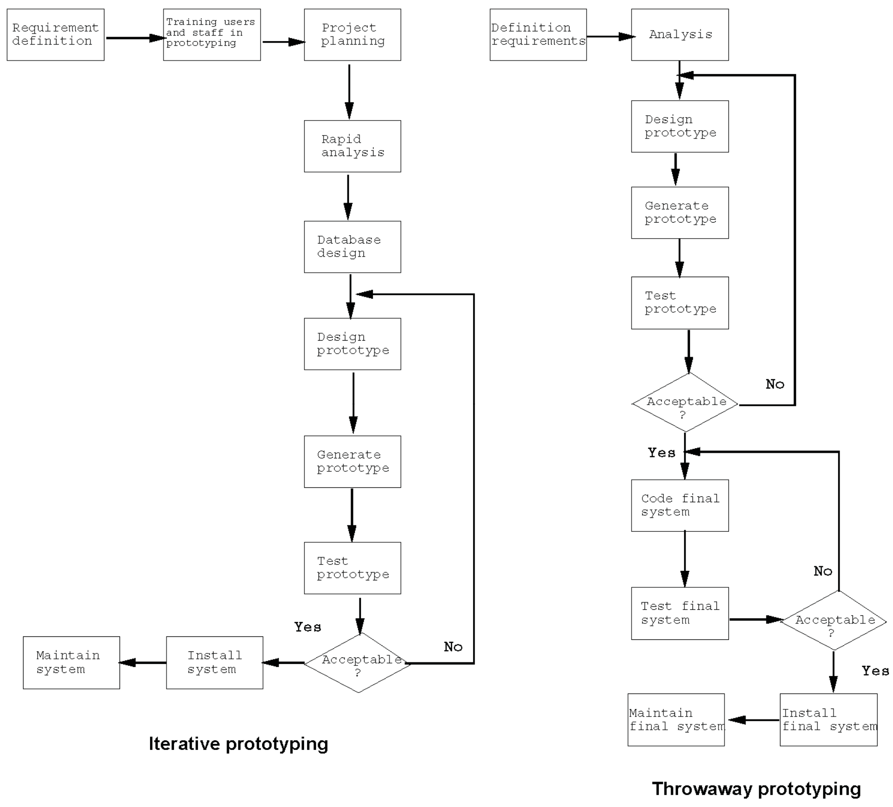

Fig. 2.5 Different forms of prototyping methodologies

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).

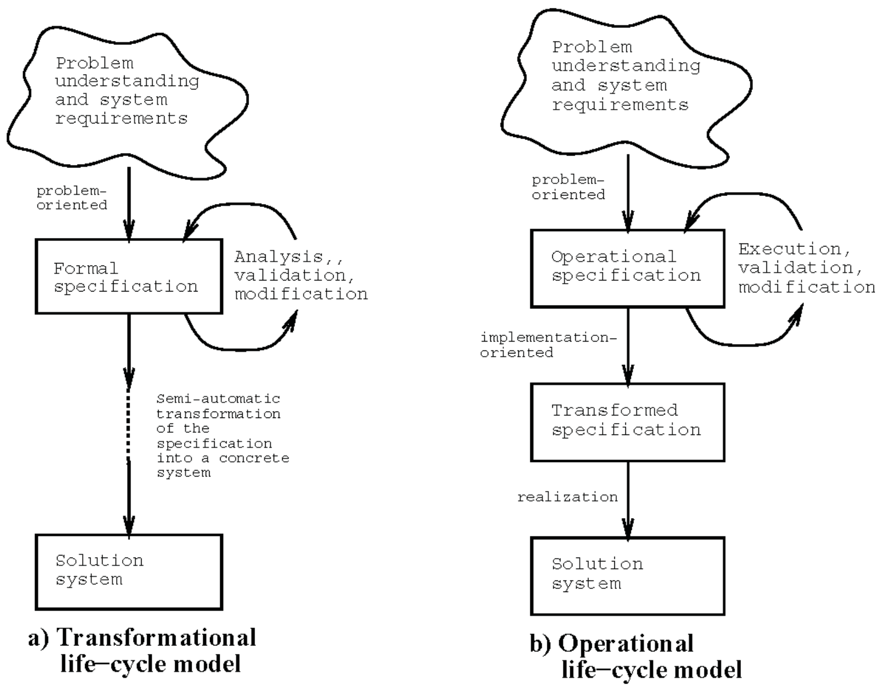
Problem understanding and system requirements
problem-oriented
Formal specification
Analysis,, validation, modification
Semi-automatic transformation of the specification into a concrete system
Solution system
a) Transformational life–cycle model
Problem understanding and system requirements
problem-oriented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

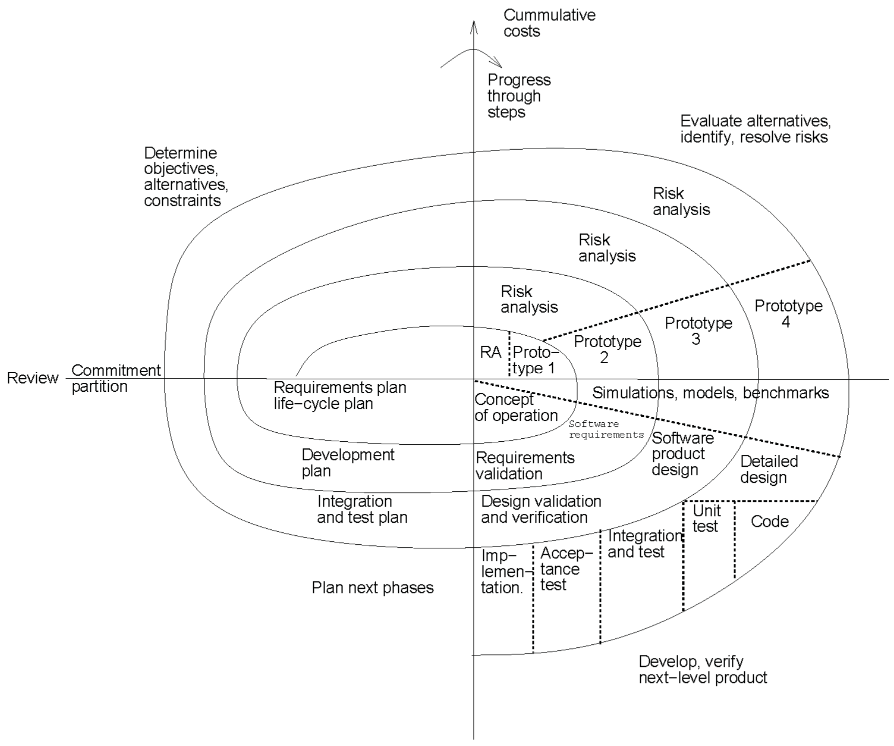

Fig. 2.7 The spiral model (adapted from (Boehm 1988))

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.

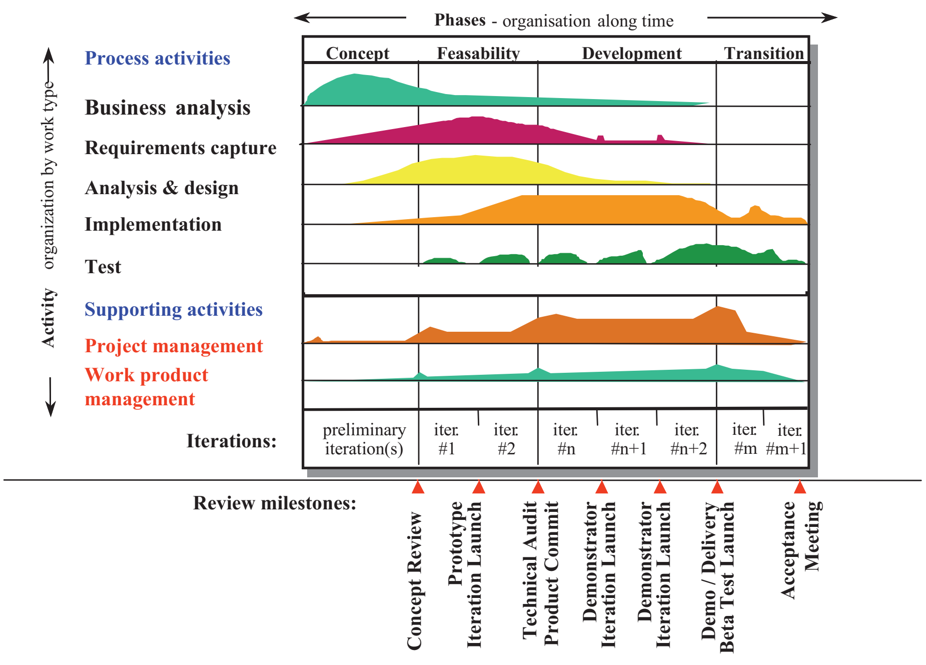

Fig. 2.8 The activity view of a development process following RUP

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)

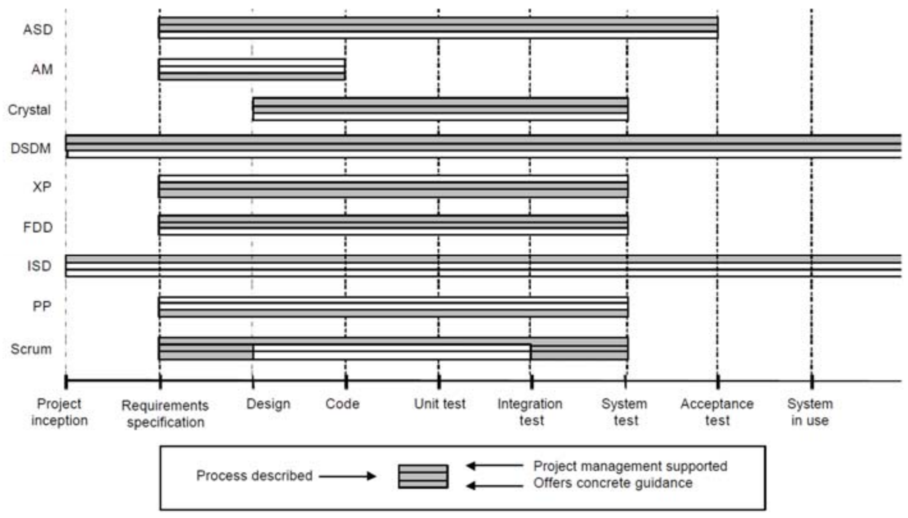

Fig. 2.9 Comparing project management, life-cycle and guidance support in agile development methods (From (Abrahamsson et al. 2010))

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:

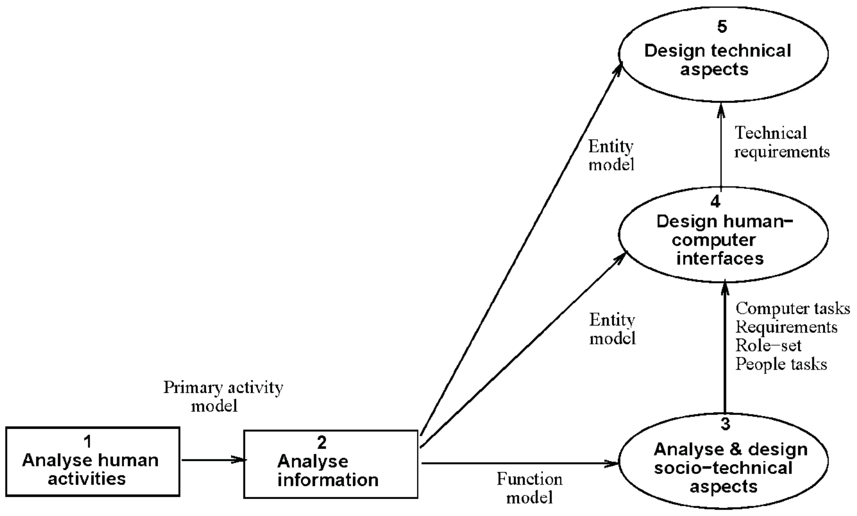

Fig. 2.10 The phases and interdependencies in Multiview

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

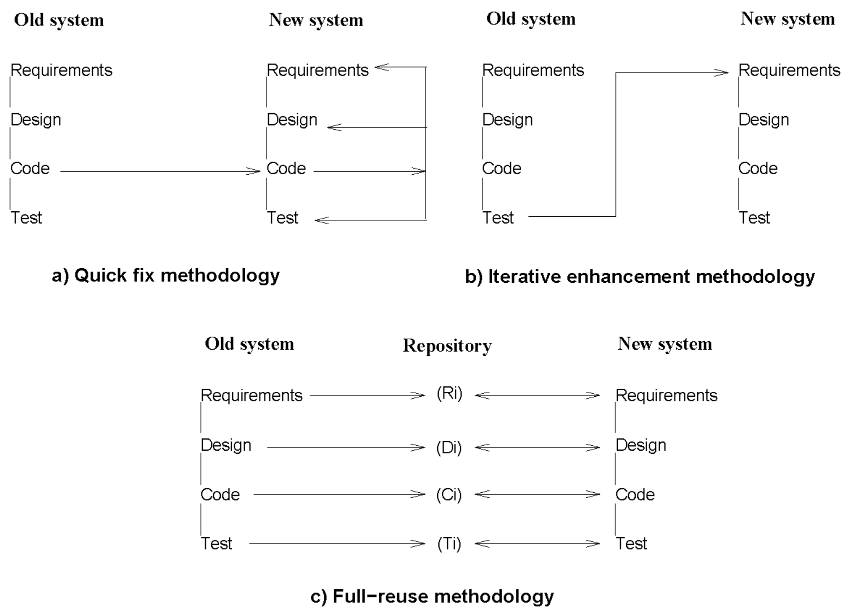

Fig. 2.11 Maintenance process models

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.

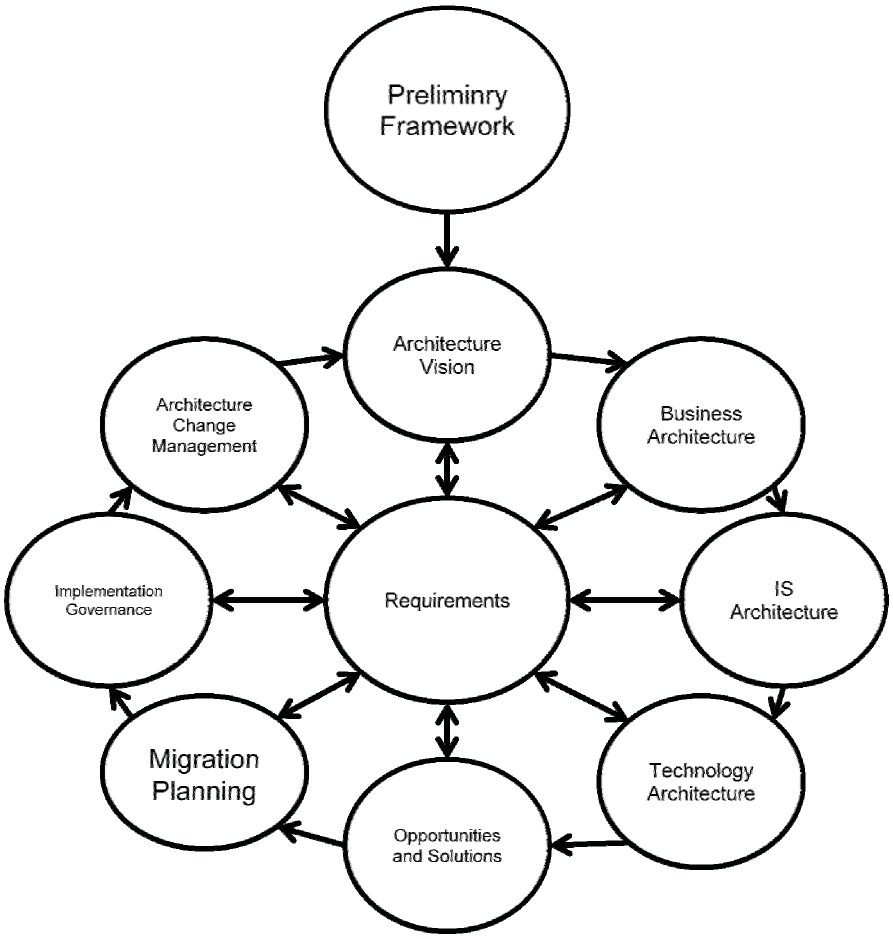

Fig. 2.12 Main steps in a TOGAF architecture development project

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.

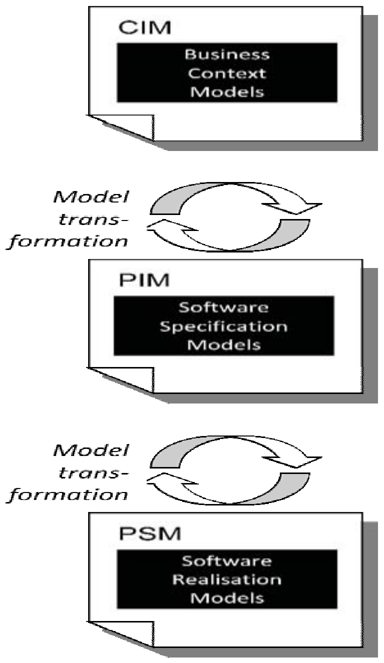

Fig. 2.13 Levels of models in MDA

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.

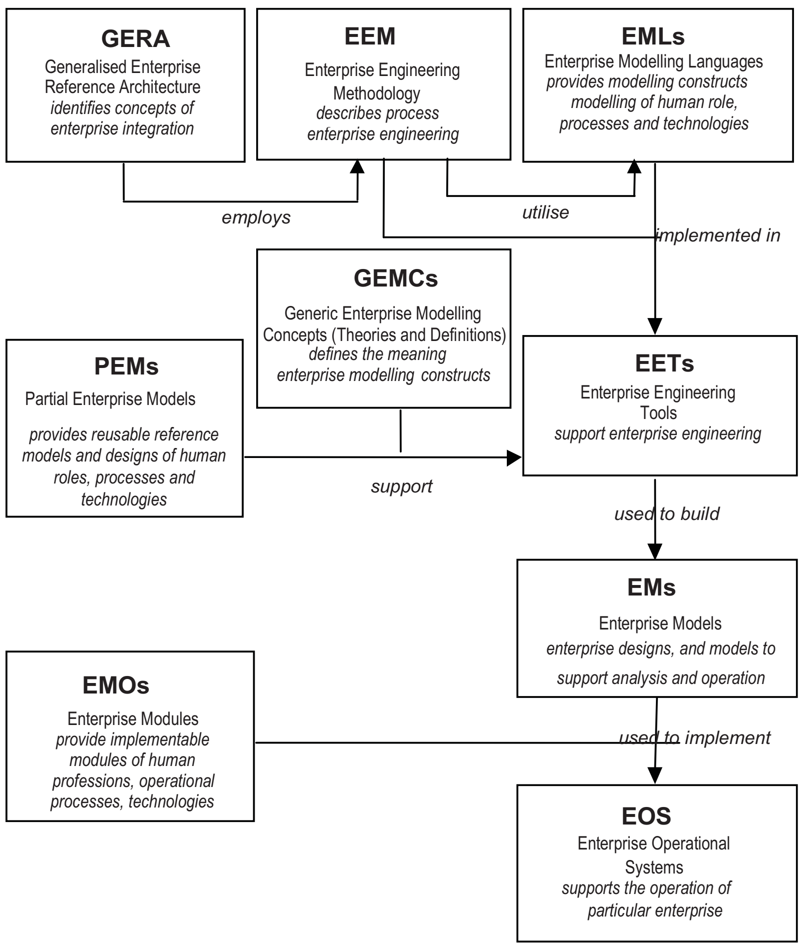
GERA
Generalised Enterprise Reference Architecture
identifies concepts of enterprise integration
EEM
Enterprise Engineering Methodology
describes process enterprise engineering
EMLs
Enterprise Modelling Languages
provides modelling constructs modelling of human role, processes and technologies
employs
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. View *Dimension:* providing for the controlled visualisation of specific views of the enterprise entity.

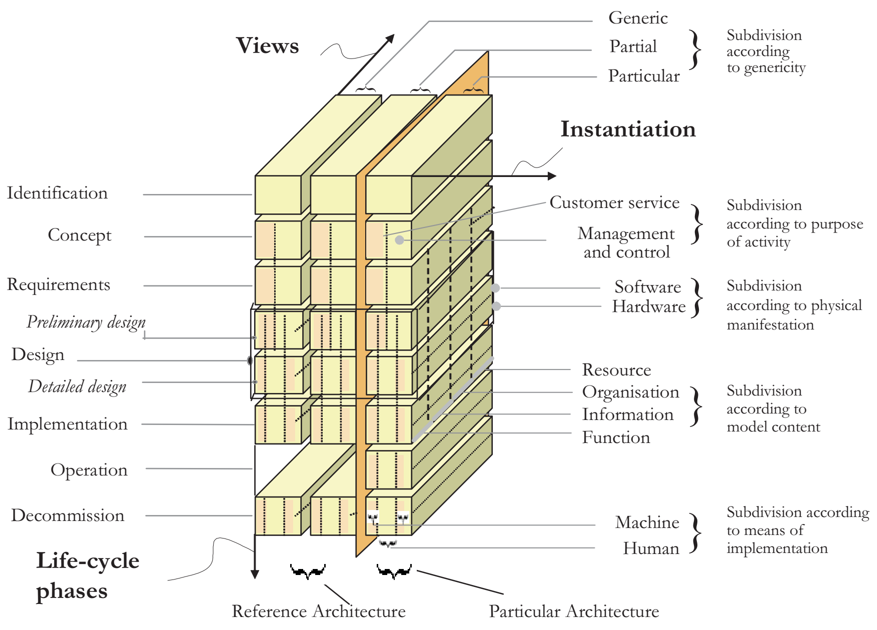

Fig.2.15 GERA Modelling Framework with Modelling Views

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 sensemaking 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.

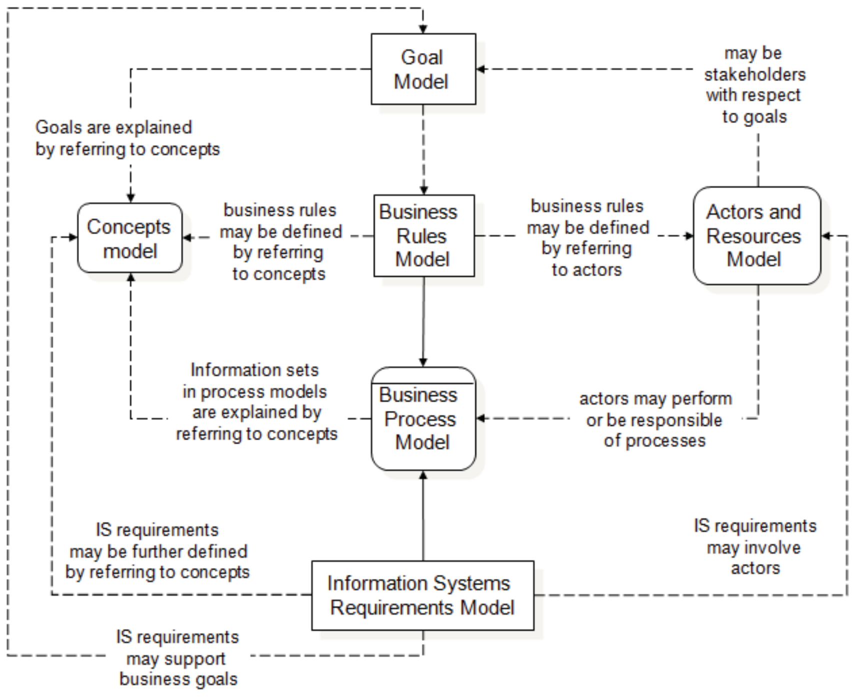
Goal Model
may be stakeholders with respect to goals
Goals are explained by referring to concepts
Concepts model
business rules may be defined by referring to concepts
Business Rules Model
business rules may be defined by referring to actors
Actors and Resources Model
Information sets in process models 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
Information Systems Requirements Model
IS requirements may involve actors
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).

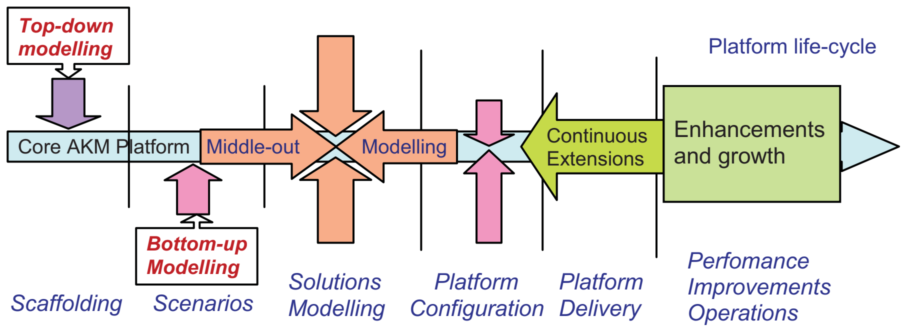

Fig. 2.17 The steps of the customer delivery process (From (Lillehagen and Krogstie 2008))

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.