Engineering Context Aware Socio-Technical Systems through Capabilities

Gonçalo Antunes
Lisbon, Portugal
goncalo.antunes@ist.utl.pt

Abstract

In the approach traditionally followed by engineering methods, systems are a product of a development process that starts with requirements gathering and analysis process, and as such, the final product is expected to be in accordance with agreed requirements. However, reality is often different and more complex, involving sometimes the adoption and integration of off-the-shelf solutions into complex systems, or even the reutilization of existing solutions in different settings than those for which they were conceived initially. Reusing of-the-shelf components motivated by only previously identified requirements they are perceived to support, while not considering the full potential of these components, might result in a dumb down usage of the existing technology, which might work as a blindfold to innovation. The concept of capability is a powerful concept adopted by many areas, as it can be used to provide a holistic view of a product or system, while offering new ways of dealing with complexity. This work proposes the development of a method for capability-based engineering of systems that also allows the combination of systems in order to deliver new types of capabilities, along with the identification of the appropriate conditions for its application. Techniques to define viewpoints that make possible the description of systems and systems’ components by the means of their engineered capabilities will also be a target.

Keywords: capability; system; methods; systems engineering.

Contents

1 Introduction
1.1 Problem ............................................. 2
1.2 Research Questions .................................. 3
1.3 Expected Contributions ............................... 4
1.4 Validation of the Research ........................... 4
1.5 Work plan .......................................... 5
1.6 Outline ........................................... 5

2 Related Concepts ..................................... 6
2.1 The Concept of Process, Activity, Technique, Method, and Methodology ............. 6
2.2 The Concept of System ................................ 6
2.3 The Concept of Architecture .......................... 7
2.3.1 In Software and Systems Engineering .......... 7
2.3.2 In Enterprise Architecture ....................... 8
2.4 The Concept of System’s Life Cycle ................. 8

3 The Concept of Capability .......................... 10
3.1 In Economics and Strategic Management .......... 10
3.2 In Systems and Software Engineering .............. 11
3.3 In Information Systems ............................. 12
3.4 Summary .......................................... 13

4 Modeling Frameworks ............................... 13
4.1 The Reference Model for Open Distributed Processing ......................... 13
4.2 ArchiMate ......................................... 14
4.3 The Open Group Architecture Framework ........ 14
4.4 The US Department of Defense Architecture Framework ......................... 15
4.5 The UK Ministry of Defence Architecture Framework ......................... 16
4.6 Enterprise Ontology ................................ 16
4.7 The Business Motivation Model ...................... 18
4.8 Summary .......................................... 19

5 Methods for Engineering Systems ................ 19
5.1 System Development Life Cycle Models .......... 19
5.1.1 The Waterfall model ............................ 19
5.1.2 The V-model .................................... 20
5.1.3 The Spiral model ................................ 20
5.1.4 The Incremental model ......................... 21
5.1.5 The SIMILAR process model .................... 21
5.2 Software Engineering Methods .................... 22
5.2.1 The Cleanroom method .......................... 22
5.2.2 The Rational Unified Process .................... 22
5.2.3 The Enterprise Unified Process ................. 23
5.2.4 The ICONIX method ............................ 24
5.2.5 The Method for an Integrated Knowledge Environment 2.0 ................. 24
5.2.6 The Dynamic Systems Development Method ....... 25
5.2.7 The eXtreme Programming method ............ 25
5.2.8 The Adaptive Software Development method ....... 26
5.2.9 The Feature Driven Development method ...... 26
5.2.10 The Scrum method .............................. 27
5.3 Enterprise Architecture Methods .................. 27
5.3.1 The TOGAF Architecture Development Method .......... 28
5.3.2 The DoDAF 6-Step Architecture Development Process ......................... 28
5.3.3 The Federal Enterprise Architecture Framework method ................. 29
5.3.4 The Design and Engineering Methodology for Organizations ................. 29
5.4 Systems Engineering Methods ..................... 30
5.4.1 The Vitech MBSE Method ........................ 30
5.4.2 The Rational Unified Process for Systems Engineering ................. 30
5.4.3 The Object-Oriented Systems Engineering Method ......................... 31
1 Introduction

The concept of capability can be defined as “the quality or state of being capable” [1] or “the power or ability to do something” [2]. Although simple, it is a powerful concept, as it can be used to provide an abstract, high-level view of a product or system, offering new ways of dealing with complexity. As such, it has been widely adopted in many areas. This work intends to explore the full potential of this concept in the engineering of socio-technical systems, information systems in particular, by innovating in the way it can be used to cope with the demands of the dynamic environments where these systems are embedded.

In economics and strategic management, capabilities are a part of the resource-based view of the organization [3, 4], which built upon the idea that firms could have the same resource inputs available but they could differ on the capability to use those resources in the most productive way [5]. In that sense, capabilities can be seen as a factor of competitive advantage which differentiates firms [6], functioning as a means for organizations to adapt better than others to changing environmental conditions [7, 8]. The concept of capability involves routines that are executed by the organization in a repeatable and often non-conscious way [8]. Building on this, the dynamic capability view states that these routines can be then changed by dynamic capabilities which allow the reconfiguration of the routines and used resources in order to adapt to changing conditions [9].

In the area of system engineering, capabilities are seen as a core concept [10]. Particularly in the military field, a capability is seen as the ability to achieve a determined military objective [11], requiring a combination of people, process and material [12]. In general, this notion is considered particularly relevant for the engineering of complex systems-of-systems (SoS), which relies on the combination of different systems for achieving a particular capability [13, 14].

In software engineering, the first capability maturity model (CMM) was developed with sponsorship from the US Department of Defense with the aim of assessing the capability of software contractors [15, 16]. For that purpose, it considers processes as capabilities, thus defining how to assess specific qualities of software engineering processes, providing a way to understand the current state of potentially complex software systems. The CMM has ever since been evolving and integrating several other models. Nowadays, it covers acquisition, development and delivery processes [17, 18, 19].

The concept was also embraced by the information systems field, namely by the Enterprise Architecture (EA) domain through its use, for example, on the US Department of Defense Architecture Framework (DoDAF) [20] and the UK Ministry of Defense Architecture Framework (MODAF) [21]. It has since then been also adopted by The Open Group Architecture Framework (TOGAF), a generic enterprise architecture framework where it is an integral part of the architectural practices described in its specification [22]. In that sense, the concept of capability has been adopted by several methodological frameworks as a mean of promoting the alignment between the strategy and goals of the organization and the technological implementation and operational skills of the individuals that compose it [22]. The concept of capability can also be found in the SoaML language [23] for the modeling of Service Oriented Architectures (SOA), with the purpose of specifying the abilities that a determined participant in a SOA needs to hold in order to use or provide a service, depending on his role.

Finally, in the field of organizational design and engineering, which advocates the combination of organization theory and engineering practice in order to create computer-based artifacts that sustain economically relevant knowledge, capabilities and their underlying routines also are considered concepts of major research importance [24]. They are considered to provide a means of observing the drivers that underlie change in organizations and, according to that, steer the organization in the right direction.

1.1 Problem

In the approach traditionally followed by engineering methods, systems are a product of a development process that starts with a requirements gathering and analysis process, and as such, the final product is expected to be in accordance with agreed requirements. This is a perfectly valid and coherent approach for systems
engineered from scratch and probably will be valid in most of the scenarios, even when a manageable number of external or off-the-shelf components are to be integrated into a product that is in line with the requirements specification. However, reality is often different and more complex, involving sometimes the adoption and integration of off-the-shelf solutions into complex systems, or even the reutilization of existing solutions in different settings than those for which they were conceived initially. In these scenarios what we really have are systems-of-systems, where the usage of a traditional development life cycle might not be the most useful, since in fact the final product is not being engineered from scratch. Also, reusing of-the-shelf components motivated by only previously identified requirements they are perceived to support, while not considering the full potential of these components, might result in a dumb down usage of the existing technology, which might work as a blindfold to innovation.

These issues are aggravated by the fact that many engineering problems are situated in complex socio-technical contexts: hardware and software systems are increasingly an important part of large-scale engineered systems of systems [25], where systems are characterized as being highly contextual and uncertainty is a constant [26]. Most of the technology-based systems inhabit an environment where human factors cannot be ignored. Large technical systems (e.g., information systems) can be considered a web of social and technical elements and history shows that ignoring socio factors often leads to failures or disasters [27]. This creates new challenges to the particular area of information systems, and on its responsiveness to the general area of systems engineering. The needed consensus concerning the mission and architecture of the information system is much difficult to attain with dynamic changes in operational and contextual matters [28]. The concept of capability can play an important role in tackling the challenges created by these issues, since it could allow dealing with architecture alignment and traceability issues at a higher level of abstraction.

Despite its wide adoption, the concept of capability is recent, which is believed still has much space for innovative approaches, as demonstrated in the working propositions described in [24]. These explicitly involve the exploration of the concept of capability and other associated concepts, such as the concept of routine. Current approaches still lack the explicit modeling of contextual concerns and the traceability down to more operational and implementation concepts.

For example, TOGAF provides the concept of capability on its meta-model, but no explicit relationships to other concepts are contained in the meta-model, but it is not clear how to determine which capabilities are needed or if the capabilities already possessed are sufficient for achieving the goals of the organization, since the modeling of the capabilities does not take into account the explicit modeling of contextual concerns that have a decisive influence on the capabilities - for that, strategic-level concepts should be present at the level of the meta-model. This fact is even more aggravated in the case of other system’s development methods that do not even take into account the concept of capability and the explicit modeling of the strategic aspects behind the development of a system.

Concluding, we believe the concept of capability seems somewhat neglected by methodological frameworks for systems development, despite its potential relevance. Therefore, although the concept of capability has been adopted by several methods, its real potential is still to be explored, namely its usage for characterizing a system and its supposed usefulness to deal with change, which motivate to these main issues to be tackled:

- **I1. System context and capabilities:** Current methods and techniques do not motivate or support the explicit modeling of viewpoints relating the system’s capabilities with the system’s environmental drivers of change.

- **I2. System architecture and capabilities:** Current methods and techniques do not motivate or support the explicit modeling of viewpoints relating the system’s capabilities with engineered system’s behavior and structural elements.

### 1.2 Research Questions

To face the pointed issues, this work proposes to explore the following generic research question:

- **How can viewpoints motivated by capabilities be effectively used in the engineering of systems, especially when these are situated in complex or unstable systems-of-systems contexts?**

This generic research question can be detailed in the following ones:

- **RQ1. Capabilities and Engineering:** How can the concept of capability be used to more effectively engineer systems aligned with system’s capabilities?
with stakeholder’s concerns? This question is related with the usage of the concept of capability in architecture viewpoints (and as such in the meta-model) and its relation to other concepts depicted in other viewpoints addressing both strategic and implementation concerns. It is expected that the concept can make the bridge between these viewpoints, helping to better deal with changes and their propagation on the architecture.

- RQ2. Capabilities and Assessment: How can the concept of capability be effectively used to support the assessment of the alignment and compliance of the systems? The concept of capability has been used before in the assessment of processes in the software engineering area. This question is related with the usage of capability viewpoints for the assessment of the compliance of systems with requirements of any nature.

- RQ3. Capabilities and Governance: How can the concept of capability be effectively used in the governance of systems, throughout the overall systems’ life cycle? This question is related with the usage of capability viewpoints and assessment results in the governance processes to ensure that the systems are aligned with their objectives throughout their life cycle.

1.3 Expected Contributions

The objectives of this work are to research and develop the fundamental techniques and methodological principles in order to allow the assessment of current systems capabilities and the design of desired capabilities taking into account contextual concerns in complex environments (including the ability to trace a system’s capabilities with the system’s expressed requirements and goals), and the effective engineering of systems that are able to cope with changes to the environment (comprising not only its development but all the life cycle of the system). For that purpose, we propose to define the concept of capability of a system, which we survey definitions of the concept with origin in different areas in section 3.

The result of this work will be the development of a method for capability-based engineering of systems (meaning conceiving a system and managing its life cycle according to its desired capabilities, which might have a relation with the requirements engineering methods) that also allows the combination of systems in order to deliver new types of capabilities, along with the identification of the appropriate conditions for its application. This method will be aligned with the main methodological frameworks, of which a survey is presented in section 6 and will be designed according to method engineering practices, which we survey in section 6. The definition of the principles to define relevant viewpoints that make possible the description of systems (and systems’ components) by the means of their capabilities will also be a target. This can occur in two scenarios: either engineered according to specific requirements, or identified in spite of the requirements that were in the foundations of the system conception. The techniques to be developed will be in line with relevant architecture frameworks, which we survey in [30] and might rely on the extension of existing business process modeling techniques (for instance, the ones surveyed in [29]).

This work intends to innovate on the techniques and methodological approaches to the engineering of systems that truly takes into account the context and its effect on the evolution of the system, namely at the level of the architecture modeling. The design and engineering of complex socio-technical systems (of which information systems are an example) at the level of the concepts on which this work focuses is seen as a means of steering the evolution of these systems. In that sense, the expected innovation rests on the fact that by making use of these concepts, a bridge will be created between the drivers of change and the requirements of a system, and thus change will be more easily propagated through the architecture.

1.4 Validation of the Research

The validation of this work will be done in the context of these possible scenarios:

- Scenario 1. the Portuguese Air Force, which has an interest in the engineering and governance of military capabilities. This scenario would fulfill the purpose of applying methods for modeling and combining the current capabilities of the different systems. This is a classical systems engineering scenario, involving the integration of technical and socio-technical systems, with the focus being on the processes.

- Scenario 2. the Informatics Center of the Instituto Superior Técnico (CIIST), which is responsible for the development of several solutions for managing academical processes. The systems developed by CI-IST are developed in an agile and incremental fashion, with functionalities being added as they are needed, in a quite ad-hoc fashion. This scenario would fulfill the

---

1 Architecture should guide the design and evolution of a system. As such, a solution to the problems described in this proposal should naturally focus on architecture.
purpose of characterizing existing systems developed by CIIST using capabilities, so that the design of new systems and incorporation of new capabilities in new systems is improved. This is a classical information systems scenario, involving a multi-technological environment with the integration of different platforms. The focus of this scenario is on the technology.

Scenario 3. the Laboratório Nacional de Engenharia Civil (LNEC) and the Laboratório de Instrumentação e Física Experimental de Partículas (LIP). LNEC is responsible for monitoring several of the larger civil engineering structures in the country for the purposes of ensuring the structural safety. For that purpose, it maintains a large repository of observation data for which it has the legal obligation of maintaining and preserving. That observational data is unique and unrepeatable, which makes its preservation crucial. LIP participates in high-profile experiments which are produced in the context of international collaborations. The results of the analysis performed by LIP’s researchers on that data are difficult to repeat due to the high complexity of the analysis, making its preservation crucial. Although several requirements specifications exist for evaluating the current abilities with respect to preservation, the way these are formulated makes the evaluation of current systems terribly hard. One way of approaching this scenario would be to try to express those requirements specifications as capabilities, so that the evaluation of current systems is enhanced, and LNEC can better deal with the problem. The focus of this scenario is on eScience data and its processing.

1.5 Work plan

The work plan to be followed until the conclusion of the dissertation is the following:

- Task 1 (until April 2012) - Preparation and presentation of the PhD proposal

- Task 2 (April 2012 - March 2013) - Research, development, and validation of the results. This will involve the following sub-tasks:
  1. (April 2012 - August 2012) Focus on defining the concept of capability and understand how capabilities can be modeled at the level of the architecture, including the relationships with drivers and constraints prompted by the environment, and the relationships with the field of requirements management. The survey on usages of the concept of capability presented in section 3 is related with this sub-task.
  2. (May 2012 - August 2012) Elaborate a meta-model that unifies contextual and operational concerns, using the concept of capabilities as a bridge. The survey on modeling frameworks presented in section 4 is related with this sub-task.
  3. (September 2012 - November 2012) Identify and analyze the most relevant systems engineering methods to determine their adequacy for the adoption of the capability paradigm. The survey on methods for the engineering of systems presented in section 5 is related with this sub-task.
  4. (December 2012 - March 2013) Development of methods for engineering systems based on the capability approach and techniques for identifying and describing the current capabilities of a system. The survey on method engineering presented in section 6 is related with this sub-task.

- Task 3 (January 2013 - October 2013) - Elaboration of the dissertation

1.6 Outline

The outline of this work is the following. Section 2 describes the concepts that provide the foundations of this work and aid on the understanding of the subjects discussed in here, such as the concept of system, method, methodology, technique, architecture, life cycle, life cycle stages, and life cycle processes. Section 3 describes the concept of capability, which is the core concept to this work, and its usage in different areas: economics and strategic management, systems and software engineering, and information systems. Due to the central role of system’s modeling in this work, relevant modeling frameworks are described and analyzed in section 4. In section 5 relevant methods for the engineering of systems are depicted, due to this work’s emphasis on methodology. For the same reasons, section 6 describes relevant process meta-models for method engineering, situational method engineering, and enterprise architecture management patterns. Finally, we conclude in section 7.

2 Related Concepts

This section describes the main concepts discussed in this work: system, method, methodology, technique, architecture, life cycle, life cycle stages, and life cycle processes. The concept of capability is described in its own dedicated section 3. Figure 1 depicts a concept
map relating those concepts, which are described in detail below.

2.1 The Concept of Process, Activity, Technique, Method, and Methodology

To understand the concept of method, it is important to distinguish it from the concept of methodology, since both terms are often used interchangeably (for instance, in \[31\]). Formally, the correct meaning for methodology is the study of methods \[31, 32\]. However, this term is often used incorrectly, adopting a similar meaning to that of method. In turn, a method is defined as a structured set of guidelines or procedures for solving or relieving problems \[33\], a family of coherent collections of methods to deal with several system problems pouring from the conceptual framework used \[34\], or a body of methods, rules, and postulates used by practitioners to research, understand, and address systems along with their questions and problems \[35\].

A method is not a technique, but applying a method might involve the use of techniques, coupled with a set of activities and the respective correspondence relating the activities and the techniques \[33\]. A technique can be described as a “defined systematic procedure employed by a human resource to perform an activity to produce a product or a result or deliver a service, and that might employ one or more tools” \[36\]. The adequacy of a technique is determined by the method itself \[37\]. A system development method governs the technical development and management subprocesses, encompassing all the activities, techniques and tools prescribed by the method \[31\]. In this context, a process is defined as a “set of interrelated or interacting activities which transforms inputs into outputs” \[38\]. Finally, in \[36\], an activity can is defined as either a “set of cohesive tasks of a process”, a “component of work performed during the course of a project”, or as a “defined body of work to be performed, including its required input information and output information”.

The term methodology might be used punctually on this work instead of the term method, yet with the same meaning, whenever authors of the works presented hereby explicitly use it.

2.2 The Concept of System

A system can be defined as a “combination of interacting elements organized to achieve one or more stated purposes” \[39\], as a “collection of components organized to accomplish a specific function or set of functions” \[30\], or as “an interdependent group of people, objects, and procedures constituted to achieve defined objectives or some operational role by performing specified functions” \[40\], or even as a “conceptual entity defined by its boundaries” \[41\].

Two viewpoints can be adopted when observing systems: white-box or black-box \[42\]. A white-box view-
point captures the construction and operation of a system, the structuring and interaction between its components (or subsystems), and is normally used for the purpose of building or changing a system. In turn, a black-box viewpoint concerns the function or usage of the system. It has no direct relation with the construction and operation of the system, and its focus is on capturing the interactions between the system and the surrounding environment.

The two viewpoints complement each other in the sense that the alternate usage of white-box/black-box allows the observer to abstract or to focus on the details of a system or its components, taking advantage of the hierarchical structure of systems. When looking at a system using a white-box viewpoint, we abstract from the details of how the subsystems are structured internally. Thus, the components or subsystems are black-boxes in the white-box view that we are adopting. We can then use a white-box viewpoint if we want to know the internal details of a component, descending a step further in the hierarchy. This can be repeated up to a point in the hierarchy where no more details are known about the internals of components, ending up with a set of black-boxes of which we only might know, in the best case scenario, the inputs and outputs.

2.3 The Concept of Architecture

The design of the architecture of a system and its components is obviously a process of major importance in its life cycle, since it makes the bridge between the requirements of the stakeholders and the implementation of the system.

2.3.1 In Software and Systems Engineering

According to the ISO/IEC FDIS 42010, Architecture can be defined as “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”.

An architecture must reflect the different viewpoints of all the interested parts, so that it can be communicated efficiently. A stakeholder is defined as a viewer that perceives and conceives the universe, using his/her senses, in order to produce conceptions resulting from the interpretation of what is observed. A viewer can form a representation of the conceptions he/she makes using a determined language to express himself. While observing the universe, a viewer will be interested in a specific part of the universe, also called a concern, and might zoom-in to that part of its conception of the universe, also called a domain. The process of abstracting a domain in a model is called modeling. In order to start a modeling process, a viewer must first construct a meta-model, comprising the meta-concepts and modeling approach, when modeling a domain.

Figure 2 provides an informative summary of the key concepts introduced by the recommended practice and their inter-relationships. The figure presents these concepts in the context of an architecture for a particular system and an associated architectural description. The recommended practice describes that a system is situated in an environment, and that it can have several stakeholders with several concerns which define the purpose of that system. Moreover, the system has an architecture which is expressed by an architecture description, providing a rationale for the architecture. The architecture description identifies the stakeholders of the system, which have concerns about the system. In turn, an architecture description may be composed of several views (which might include several models of the architecture), which are according to the viewpoint of the stakeholder (which is used to cover the concerns of the stakeholder). The viewpoints can be composed of several kinds of models.

Viewpoint refers to a pattern or template for representing one set of concerns relative to an architecture, while a view is the actual representation of a particular system. A viewpoint provides the formalization of the groupings of models. A view is a representation or description of the entire system from a single perspective. In contrast to a viewpoint, a view refers to a particular architecture of a system. A view is primarily composed of models, although it also has additional attributes. The models provide the specific description,
2.3.2 In Enterprise Architecture

The related field of Enterprise Architecture has the purpose of modeling the role of information systems and technology on the organization, aligning the enterprise-wide concepts, aligning the business processes and information with the information systems, planning for change, and providing self-awareness to the organization, in an holistic approach to system’s architecture [46]. It provides a coherent whole of architecture principles, methods, and models that are used in the holistic design and realization of an enterprise organizational structure, business processes, information systems, and infrastructure [47]. Enterprise architecture practice involves the use of enterprise architecture frameworks that guide the architecture work, providing an underlying structure for architecture descriptions.

The Zachman Framework was one of the first enterprise architecture frameworks created. It is considered a means for defining the role of information systems in the enterprise, with the purpose of providing a holistic view of the organization functioning as a “classification theory about the nature of the enterprise”, and presenting the “kinds of entities that exist within” [2].

Figure 3 depicts the Zachman framework. It is a table where each cell can be related to a set of models, principles, services, and standards needed to address the concerns of one or more stakeholders. The rows depict the viewpoints of the stakeholders of an organization on the organization itself: Scope, which defines the business context, including the business purpose and strategy; Business Model, which describes the organization; System Model, which describes how the systems will satisfy the organization’s information needs, in a way independent from implementation; Technology Model, which describes the implementation of the systems; Components, which details each of the system’s components before production; and Instances, which gives a view of the functioning system in its operational environment.

The columns express different perspectives on each of the viewpoints: Data/What, refers to the information and data objects of the organization; Function/How, describes the functioning of the organization and its systems; Network/Where, refers to spacial elements and their relationships; People/Who, refers to the actors of the organization and of its systems; Time/When, refers to timing and events; and Motivation/Why, refers to the overall motivation, rules and constraints to the objectives.

Although it makes suggestions on the types of models/contents that might occupy each of the cells, the Zachman Framework does not make prescriptions on that matter, or even on the relationships existing between the cells.

2.4 The Concept of System’s Life Cycle

Every system, natural or created by man, can be thought of having a life cycle. A system life cycle is a conceptualization expressed by phases or stages that represent the evolution of a system, covering from its inception until its disposal. A set of ISO standards define terminology and conceptualizations of the life cycle of a system and the processes supporting the management of the life cycle. In ISO/IEC TR 24748 [49], the following life cycle stages are suggested:

- Concept stage, which deals with identifying and refining stakeholders needs and explore possible solutions;
- Development stage, in which system requirements are refined and the solution is designed, verified, and tested;
- Production stage, where the system is produced;
- Utilization stage, where the system is operated;
- Support stage, which deals with the maintenance of the system, and might occur at the same time as the Utilization stage;
- and the Retirement stage, which is related with the disposal of the system.

These stages can only be considered a suggestion, since different conceptualizations of the life cycle can exist, depending on the nature and purpose of the system, and even on the specific environment that the system is going to inhabit. Several methods to be described in the following sections contain their own conceptualization of the life cycle.
Table 1: Technical processes in a system life cycle according to ISO/IEC Std. 15288:2008 [39]

<table>
<thead>
<tr>
<th>Process</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholder requirements definition</td>
<td>Define the requirements of the stakeholders, through the identification of the stakeholders and their needs.</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>Transform the requirements of the stakeholders into system (technical) requirements which can then be used in the design of the system.</td>
</tr>
<tr>
<td>Architectural design</td>
<td>Synthesize a solution that is according to the requirements of the stakeholders.</td>
</tr>
<tr>
<td>Implementation</td>
<td>Realization of the system according to the requirements of the stakeholders.</td>
</tr>
<tr>
<td>Integration</td>
<td>Combination of different system elements in order to comply with the requirements.</td>
</tr>
<tr>
<td>Verification</td>
<td>Attest if all the requirements are being fulfilled by the system.</td>
</tr>
<tr>
<td>Transition</td>
<td>Transfer of custody of the system into the customer premises and all necessary activities needed to ensure that the customer is ready to operate the system.</td>
</tr>
<tr>
<td>Validation</td>
<td>Validation of the stakeholder requirements during the stakeholder requirements definition process and during the transition process.</td>
</tr>
<tr>
<td>Operation</td>
<td>Usage of the system.</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Sustainability of the system regarding its capacity to deliver the services for which it was designed.</td>
</tr>
<tr>
<td>Disposal</td>
<td>Deactivation of the system.</td>
</tr>
</tbody>
</table>

In turn, during the life cycle of a system a series of processes might be executed during one or even several stages, supporting its evolution. Some of the processes might be specific to a stage, or their execution might span different stages of the life cycle. In ISO/IEC Std. 15288:2008 [39], four groupings of processes are described: technical processes, project processes, agreement processes, and organizational-enabling processes.

Technical processes are processes that directly support each of the life cycle phases. Project processes concern the management of projects, including the establishment and evolution of plans, assessing the progress of projects, and control the execution of projects. Agreement processes concern the establishment of agreements between organizations, namely the acquisition and supply processes. Organizational project-enabling processes ensure that the organization has the resources and infrastructure necessary to support projects and ensure the satisfaction of objectives and agreements. These are described respectively in Tables 1, 2, 3, and 4.

Due to the characteristics of the software engineering domain, a specialization of the life cycle processes described above exists. In ISO/IEC Std. 12207:2008 [50] specializations of the processes described above for software-based systems as well as additional processes for supporting the life cycle of software systems are provided: software implementation processes, software support processes, and software reuse process.

Software implementation process acts as a substitute to the Implementation process described in Table 1, the processes described in Table 3 are sub-processes of this process. Software support processes provide assistance to the software implementation processes and are de-
Table 5: Software implementation processes in a system life cycle according to ISO/IEC Std. 12207:2008 [50]

<table>
<thead>
<tr>
<th>Process</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software requirements analysis</td>
<td>Aims to establish the requirements of the software elements of the system, maintaining consistency and traceability between system and software requirements.</td>
</tr>
<tr>
<td>Software architectural design</td>
<td>Design of the architecture which can be implemented and verified according to the requirements.</td>
</tr>
<tr>
<td>Software detailed design</td>
<td>Refined design of the software according to the architecture and to the requirements.</td>
</tr>
<tr>
<td>Software construction</td>
<td>Production of software units according to the design specifications.</td>
</tr>
<tr>
<td>Software integration</td>
<td>Combining the developed software units and components into a overall operational software system consistent with the design and the requirements.</td>
</tr>
<tr>
<td>Software qualification testing</td>
<td>Confirming if the final product conforms with the requirements.</td>
</tr>
</tbody>
</table>

Table 6: Software support processes in a system life cycle according to ISO/IEC Std. 12207:2008 [50]

<table>
<thead>
<tr>
<th>Process</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software documentation management</td>
<td>Developing and maintaining documentation during the execution of other processes.</td>
</tr>
<tr>
<td>Software configuration management</td>
<td>Aims to establish and maintain the integrity of the software items of a process or project, making them available for the interested parties.</td>
</tr>
<tr>
<td>Software quality assurance</td>
<td>Provide assurance that products and processes comply with plans.</td>
</tr>
<tr>
<td>Software verification</td>
<td>Confirms that the software products are according to the requirements.</td>
</tr>
<tr>
<td>Software validation</td>
<td>Confirms that the requirements of the software product are fulfilled for a specific use.</td>
</tr>
<tr>
<td>Software review</td>
<td>Maintaining a common understanding with the stakeholders of the project to ensure that the product is according to their wishes, throughout the life cycle.</td>
</tr>
<tr>
<td>Software audit</td>
<td>Determine the compliance of the product and processes with requirements, plans and agreements.</td>
</tr>
<tr>
<td>Software problem resolution</td>
<td>Ensure that all discovered problems are identified, analyzed, managed and solved.</td>
</tr>
</tbody>
</table>

3 The Concept of Capability

The concept of capability has been adopted in several areas of knowledge. In generic terms, it is related to having the ability of doing something [2]. In this sec-

Table 7: Software reuse processes in a system life cycle according to ISO/IEC Std. 12207:2008 [50]

<table>
<thead>
<tr>
<th>Process</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain engineering</td>
<td>Developing and maintaining domain models, domain architectures, and assets for the domain.</td>
</tr>
<tr>
<td>Reuse asset management</td>
<td>Manage the life of reusable software components.</td>
</tr>
<tr>
<td>Reuse program management</td>
<td>Plan, establish, manage, control, and monitor the reuse program and exploit reuse opportunities.</td>
</tr>
</tbody>
</table>

3.1 In Economics and Strategic Management

The concept of organizational capability has been used for many years in economics and strategic management essentially to explain competitive advantage. Early mentions of the term, although not reified, include the ones by Penrose [5], which indicated that the differences between firms could be explained by differences in the capabilities to deploy resources that were available to all firms, and by Richardson [51], which pointed that those differences were explained by the fact that firms tend to specialize in activities for which their capabilities offer competitive advantage.

In their 1973 publication, Nelson and Winter introduced the idea that competitive advantage comes both from the internal and external search processes for enhanced production capabilities and the “natural selection” processes that influence the growth and contraction of organizations, resulting as an indirect consequence of the search [52]. As described later by the authors, the usage of the term capability in this work came from the involvement of the authors in the military field at the time, and not directly from the works cited above [53]. The same authors later presented their evolutionary theory that described organizational capabilities as consisting of the ability to “perform and sustain a set of routines” [8]. Those routines are “habitual reactions” that involve coordination among the actors of the organization and the usage of skills, organization and technology to respond to the demands of the environment. Routines can even be considered the building blocks of capabilities, since for a set of activities and associated resources and skills to be considered a capability, there is a need for repeatability [6]. In order to survive, organization should engage on search operations which involve the evaluation of the current situation and changes to the organizational capabilities, if needed [8]. Chandler described organi-
zational capabilities as the “collective physical facilities and human skills”, “carefully coordinated and integrated”, as a means of achieving economies of scale and scope, highlighting their importance in the evolution of capitalism [7].

The concept of organizational capability promoted by Nelson and Winter was highly influential for the development of the concept of dynamic capability, initially developed by Teece and Pisano [54]. The former notion of capabilities can explain why firms attain competitive advantage in a determined market, but it fails to explain why some firms can adapt to highly disruptive changes in the environment prompted by technological change, critical timings, or even change in markets and competition [9]. Dynamic capabilities involve “reconfiguring the internal and external organizational skills, resources, and functional competences to match the requirements of a changing environment” [9]. The same work also describes the existence of factors that can be used to assess the distinctive capabilities of an organization (i.e., those which cannot be easily replicated by others), and that are classified in three categories: processes (i.e., the routines or other activities), positions (i.e., current technological infrastructure, intellectual property, customers, relation with suppliers, etc.), and paths (i.e., strategic alternatives available to the organization). The relationship between these three categories is explained by the fact that the essence of capabilities lies in the processes of the organization. However, competitive advantage is driven or constrained by the positioning of the internal and external assets of the organization and by the evolutionary path that the firm has chosen to adopt. In the words of the authors, “what a firm can do and where it can go are thus rather constrained by its positions and paths”. Those factors can only deliver competitive advantage if the capabilities are based on a collection of routines, skills, and assets that are difficult to imitate.

The work in [55] argues that dynamic capabilities are not themselves sufficient for attaining competitive advantage, since their functionality can be duplicated by organizations. However, competitive advantage lies on the resource configurations deployed by those capabilities. Additionally, dynamic capabilities are important for achieving short-term advantages through reconfiguration of the resources in order to make the most out of an opportunity. In order to be effective, dynamic capabilities often need to rely on new knowledge, which might involve experimental activities, such as prototyping, real-time information, and experimentation. The evolution of these capabilities is guided by well-known learning mechanisms. Product development routines, strategic decision making, resource allocation routines, knowledge creation routines, alliance and acquisition routines are given examples of dynamic capabilities.

In order to explain the evolution of capabilities, the work in [8] describes a generic capability life cycle framework that can be applied to any type of organizational setting. The framework divides the life cycle of a capability in three plus six stages. The three first stages are: the founding stage, which marks the “birth” of a capability; the development stage, which represents the building up of the capability; and the maturity stage, which marks the ending of the capability building. The maturity stage can then be followed by any of the following six stages in different combinations or orders (or even simultaneously in some cases): retirement, which marks the death of a capability; retraction, which depicts the gradual decline of a capability; renewal, which depicts the improvement of the level of a capability, and which might involve minor or major modifications to a capability; replication, which depicts the transfer of a capability into a new market; redeployment, which represents another type of capability transfer, this time for producing a different but closely related result; and recombination, which aims to improve a capability through the combination of existing capabilities.

3.2 In Systems and Software Engineering

The International Council on Systems Engineering (INCOSE3 defines Systems Engineering as being “an interdisciplinary approach and means to enable the realization of successful systems”, focusing in the whole life cycle of the system, from the definition of stakeholders’ requirements to the dismantlement of the system, considering the business and technical needs of customers in order to provide a quality product [56]. One of the origins of systems engineering is arguably the military field [56]. As such, much of the terminology used in the domain has been adopted from that origin, including the concept of (military) capability. A military capability is defined as the ability to achieve a determined military objective [11], requiring a combination of people, process and material [12].

It was precisely on the military field that the first capability maturity models (CMM) appeared. The Software Engineering Institution (SEI) of the Carnegie Mellon University, funded by the U.S. Department of Defense produced the first capability maturity model for assessing software engineering processes [15, 16]. The main purpose of the CMM is to achieve a controlled and measurable software engineering practice

---

3http://www.incose.org/
that can be continuously improved [15]. In this specification, software engineering capability is divided in three areas: organization and resource management, software engineering process and its management, and tools and technology. Despite the fact that this segmentation seems to match the triplet of people, process and material, the term process is used throughout the specification as a synonym for capability.

CMM was launched in its 1.0 version in 1991 [57], and version 1.1 came out in 1993 [58], incorporating the feedback from the software engineering community. Soon, capability maturity models began being adopted by other areas, including the more general area of systems engineering, with the purpose of improving its processes. In the A Systems Engineering Capability Maturity Model, Version 1.0, issued in 1994, a capability is defined as a “measure of the system’s ability to achieve the mission objectives, given that the system is dependable and suitable” and as a “systems engineering metric” [59]. Based on this work, the Electronic Industries Alliance [6] published standard EIA-731.1 - Systems Engineering Capability Model, which described capability as involving the attributes of people, technology, and process [60]. In an effort to unify capability maturity models, the SEI published the Capability Maturity Model Integration (CMMI) as a unifying model for three different process areas: acquisition, development, and services [17, 18, 19].

All these maturity models have in common the fact that the assessment of the maturity of software engineering practices results on a classification of different criteria situated in the areas mentioned above in one of five levels: 1. Initial, when no control or measurement of the process is executed; 2. Repeatable, when some repeatable initial level of measurement and control over costs and schedules is performed; 3. Defined, where the processes are well defined and understood, and aspects surrounding it can start to be improved; 4. Managed, where after the process being well defined, measurements can be made and quality improvements begin to be notorious; and 5. Optimized, where high degree of control over a process is achieved and continuous improvement is a reality.

It was also in the military field that the idea that capabilities could be used as essential building blocks in engineering efforts was formed [12]. Capability engineering is a process which supports capability management throughout the life cycle of a capability. Capability management aims to manage capabilities through an integrating framework consisting of three interrelated functions: capability generation, which refers to the conception, development, planning, acquisition and management of a capability; capability sustainment, which refers to the sustainability of a capability at an appropriate level of readiness, for a determined time horizon; and capability employment, which refers to the planning for and conducting military operations which involve the use of the capability.

The concept of capability was also adopted in IBM’s Rational Method Composer [4] more precisely the concept of capability pattern. The Method Composer allows the customization of the Rational Unified Process (RUP) for software engineering (which will be described in section 5.2.2). RUP provides several disciplines which are collections of tasks which can be applied during the life cycle of a system. These tasks can be combined into workflows. A capability pattern is a reusable process which can be applied at any stage of the life cycle and specifies a work breakdown structure (the workflow), the team allocated to the activities, and the work products produced from the activities.

### 3.3 In Information Systems

An information system can be defined as “an information processing system, together with the associated organizational resources such as human, technical, and financial resources, that provides and distributes information” [36], a definition which in some sense presents some similarities with that of capability. One of the main research topics in information systems is enterprise architecture.

Enterprise Architecture (EA) is a holistic approach to systems architecture with the purposes of modeling the role of information systems and technology on the organization, aligning the enterprise-wide concepts, aligning the business processes and information with the information systems, planning for change, and providing self-awareness to the organization [46]. Despite the fact that it was first created with a more traditional company setting in mind, its practices where also adopted by the military field with the surfacing of two well known enterprise architecture frameworks: the US Department of Defense Architecture Framework (DoDAF) [20] and the UK Ministry of Defense Architecture Framework (MODAF) [21].

Both frameworks adopt the concept of capability, and explicitly model it through its inclusion on the meta-model and on the viewpoints provided by the framework. DoDAF defines capability as being an ability to achieve a desired effect under specified conditions through the combination of activities and resources [20]. In MoDAF, a capability is defined as a a clas-

---

4 Already extinct.

5 http://www-01.ibm.com/software/awdtools/rmc/
sification of some ability that the enterprise possesses, and that it can be specified whether the enterprise is able to achieve it or not [21]. Capabilities in MoDAF can be represented through a composite structure entitled capability configuration, which we define as “a set of artefacts or an organization configured to provide a capability”, and involves physical, human, and software resources and the interactions between them.

The concept of capability is also making its cross to general enterprise architecture approaches. The Open Group Architecture Framework (TOGAF), a generic enterprise architecture framework where it is an integral part of the architectural practices described in its specification [22]. The concept is a part of the meta-model and a capability-based planning for business is included as one of the techniques provided by the specification. TOGAF defines capability as “an ability that an organization, person, or system possesses”, and that it typically “requires a combination of organization, people, processes, and technology” [22].

Another relevant area in information systems research and practice is that of modeling languages. The Business Service Description Language (BSDL) has the purpose of describing business services from a pure business perspective, addressing specifically their decomposition and non-functional properties [61]. It aims to close the gap existing between more strategy and goal description languages and operational service description languages. BSDL aims to model both functional and non-functional concepts related to business services. The concept of capability is modeled as a functional concept which represents a function that is performed by a business service. SoaML is another example of the inclusion of the concept of capability in the meta-model of the language. The usage of this concept allows the expression of service architectures in terms of the logical capabilities of the services in a way that is agnostic to participants in the architecture, identifying a set of functions or resources that a service might provide [23]. In that sense, this definition of capability is more operational than the others presented before.

3.4 Summary

From the definitions presented above it is clear that a capability involves processes, people and other resources. It is also clear that the availability of resources in the right time and place (also referred to as positions [9]), together with the strategic decisions taken (also referred as paths [9]) have a decisive influence on the effectiveness of how a capability is delivered and the competitive advantage that can be achieved concerning the similar capabilities of the competition.

Making the parallel with the five W’s and one H of communication used in the Zachman framework (What, How, Where, Who, When, and Why), it can be said that capabilities are delivered by means of the behavior of the system, which might involve the perspectives of How it is done (function), Who does it (people), and using What (resources). Those processes are constrained by contextual aspects, such as the positioning of the resources, which involves the perspective of Where (location) and When (time) are they available. Capabilities are also shaped by the organizational drivers leading to important decisions affecting the capability, which involves the perspective of Why (motivation) a decision is taken. These dimensions should provide a complete characterization of the different factors surrounding a capability. Figure 4 depicts this perspective framework.

4 Modeling Frameworks

In this section, relevant modeling frameworks are described. These frameworks aim to model different aspects of a system, ranging from organizational decision making down to hardware components.

4.1 The Reference Model for Open Distributed Processing

The Reference Model for Open Distributed Processing (RM-ODP) is an architecture framework for the development of software-based systems, with a particular concern for distribution, integration, and portability [62]. It offers a set of viewpoint specifications that can...
be used for developing views of the systems using an object-oriented modeling approach.

Five viewpoints are provided by the framework: (i) the enterprise viewpoint, which is concerned with the governance of the system, including purpose, scope and policies; (ii) the information viewpoint, which is concerned with the information manipulated by the system and all the aspects surrounding it; (iii) the computational viewpoint, which is concerned with the functional decomposition of the system and definition of component interfaces; (iv) the engineering viewpoint which is concerned with the infrastructure; and (v) the technology viewpoint, which is concerned with the concrete technology that will support the system.

RM-ODP does not make use of the capability concept. However, it allows the modeling of contextual aspects of systems through its enterprise viewpoint. These contextual aspect are also related to the goals and to operational aspects. Figure 5 depicts an excerpt of the meta-model for the enterprise viewpoint.

4.2 ArchiMate

ArchiMate is a modeling language for enterprise architecture with an already structured concepts and relationships [47]. It uses an architectural framework for structuring the concepts and relationships of the language, which is divided in three horizontal layers (Business, Application and Technology), which are further refined according to three distinct aspects (Information, Behavior and Structure).

The Business layer addresses the domains of Information, Product, Process and Organization. The Application layer addresses the domains of Application and Data. The Technology layer addresses the domain of the Technological Infrastructure. As for the aspects, the Structure aspects represent the key to interpret the layer. The Behavior aspects describe what the Structure aspects do. The Information aspects comprise the entities that are used by the Behavior aspects. A motivation extension for ArchiMate is being prepared for inclusion in the next version of the framework, addressing more strategical aspects related to the architecture [64]. Figure 6 depicts ArchiMate’s meta-model.

4.3 The Open Group Architecture Framework

The Open Group Architecture Framework (TOGAF) [22] is a high-profile EA, providing methods and tools to support architecture development. It comprises seven modules which can be partly used independently of each other. The core of TOGAF is the Architecture Development Method (ADM), which will be explained ahead in section 5.3.1 and the Architecture Content Framework.

The Architecture Content Framework is TOGAF’s own architecture framework. It specifies the different types of architecture products and provides a Content
Meta-model which can be related with the phases of the ADM. The meta-model defines the kinds of entities existing in an enterprise, at multiple levels, and the horizontal and vertical relationships existing between those entities, which could point to possible dependency relationships. The meta-model entities can then be instantiated in the development of concrete models of the organization.

As can be observed in Figure 7, the content meta-model defines the concept of capability and places it on the top-layer dealing with the Architecture Vision, requirements and the road map for the architecture. A particularity of the meta-model is that no relations are made within the entities belonging to that layer, with the exception of the relationship between capability and work package. Moreover, no relationships are defined between the concept of capability and the concepts belonging to other layers. Even the relations between the other concepts of the same layer and the concepts belonging to other layers are not clear, since no semantic is given to the relationships. The only given indication is that those concepts are associated with all other concepts of the architecture.

4.4 The US Department of Defense Architecture Framework

The United States Department of Defense Architecture Framework (DoDAF) supports the unification of the architectures of different US military commands, services and agencies to improve decision making and information sharing [20]. It provides a meta-model, defining the concepts to be represented on the architecture models, a set of viewpoints and corresponding models, and an architecture development process, which will be described in more detail in section 5.3.2.

DoDAF’s meta-model is divided in three levels: a conceptual data model, a logical data model, and a physical exchange schema. The conceptual data model contains high-level, non-technical concepts suited to managers and executives. The logical data model adds technical information to the conceptual model, describing it in an unambiguous way. Finally, the physical exchange schema consists on the logical data model generated as a .xsd, with data types and implementation attributes. Figure 8 depicts the main concepts modeled in DoDAF (Based on: [20]).

The framework also defines a set of viewpoints: the All viewpoint, which describes general aspects of the architecture; the Capability viewpoint, which is related with the strategic requirements and phasing of capability deployment and delivery (although, as shown above, these concerns are not modeled explicitly in the metamodel); the Data and Information viewpoint, which aims to depict the information requirements of business and operations; the Operational viewpoint, which is concerned with the operational perspective of capabilities; the Project viewpoint, which describes the relationships between the capabilities, operations, and the projects being implemented; the Services viewpoint, which depicts the solutions and relates these to
4.5 The UK Ministry of Defence Architecture Framework

The United Kingdom Ministry of Defence Architecture Framework (MODAF) was created in 2005 in order to support the planning and change management activities of the UK Ministry of Defence (MOD) [21]. It was originally based on DoDAF, but has since then diverged, reflecting the requirements of the MOD. Contrary to DoDAF, MODAF does not prescribe an architecture development process, yet guidelines on the application of the framework are provided. However, it does prescribe a meta-model and a set of architecture viewpoints.

The MODAF meta-model defines entities for each viewpoint. Some entities are used in several viewpoints, thus enforcing traceability. The concept of capability is defined at the level of the strategic viewpoint and it is related to enterprise goals through the vision for the architecture for a determined phase of the life of the organization. Capabilities can depend on or be composed of other capabilities, and are measurable. Capabilities are then related to operational concerns through the concept of Operational activity contributing for the capability. A operational activity might be realized by a node which is realized by a Capability Configuration which involve resources. Finally, a capability configuration is delivered by a project through capability increments. Figure 9 depicts a high-level view of the MODAF meta-model.

The each viewpoint defined by MODAF is part of one of the following sets: the All View viewpoint, which is similar to DoDAF’s All viewpoint; the Strategic View viewpoint, which is similar to DoDAF’s Capability viewpoint; the Operational View viewpoint, which can be though as a merge between DoDAF’s Data and Information viewpoint and the Operational viewpoint; the System View viewpoint, which is similar to DoDAF’s System viewpoint; the Technical Standards View viewpoint, which can be related to DoDAF’s Standards viewpoint; the Acquisition View viewpoint, which is related with the acquisition and integration of new capabilities and finally, the Service-Oriented View viewpoint, which is similar to DoDAF’s Service viewpoint.

4.6 Enterprise Ontology

Enterprise Ontology (EO) can be defined as ”the realization and implementation independent essence of an enterprise” or ”the deep structure behind its observable surface structure” [22]. In other words, Enterprise Ontology intends to capture the enterprise in a transparent way. It is supported by a theoretical background called the $\psi$-theory, which separates the function from the construction of the enterprise. The services which the enterprise provides to the environment represent the function of the enterprise, and the collective activities from which the services are delivered and the human actors which perform those activities represent the construction of the enterprise.

The $\psi$-theory states that subjects on an organization perform two kinds of acts: (i) production acts (p-acts), which contribute to produce goods or services, the successful execution of which results in production facts (p-facts); and (ii) coordination acts (c-acts), which serve the purpose of creating commitments between subject regarding the p-acts, the successful execution of which results in coordination facts (c-facts). The patterns combining c-acts and p-acts with the purpose of bringing into existence a product or service is called a transaction. A business process can then be consid-
Figure 10: The different layers of the organization (Source: [42]).

A “tree structure of enclosed patterns” [42] in which subjects perform acts, they are fulfilling actor roles.

In one of the axioms of the supporting theory, an organization is defined as the nesting of three different layers: the B-organization layer, the I-organization layer, and the D-organization layer. The B-organization is the aspect of an organization through which business services are delivered or provided by the environment. The ontological acts performed in the organization are at this level, despite other types of acts are capable of being performed at this same level. The I-organization is responsible for providing infological services to the B-organization, which are concerned with the content of data (e.g., remembering, recalling, and computing facts). The D-organization is responsible for providing datalogical services to the I-organization, which are concerned with the form of data (e.g., storage, retrieval, and transformation of facts). Figure 10 depicts the different layers of the organization.

A meta-model for the definition of the ontology of the enterprise is provided, which is called the CRISP model. The CRISP model assumes that an organization can be defined formally by a tuple <C, R, I, S, P>. Other assumption is that an organization, at each moment disposes of an agenda composed of several agendum. The components of the tuple can be defined as follows:

- C consists of a set of C-facta, called the coordination base. C-facts are created when a C-act is successfully performed by an actor. C-facts are also called agendum which corresponds to the C-fact that is going to be dealt with at the moment. A C-fact corresponds to an intention about a P-fact (which normally can be request, promise, state, or accept). An example of a C-fact can be "membership #348 has been started is requested", "requested" being the intention, and "membership #348 has been started" being the P-fact. A C-fact also has a time associated with the P-fact. In the case of the given example, the time associated with the P-fact would be the requested creation time of the P-fact.

- R consists of a set of action rules, called the rule base. Action rules are rules executed by the actor roles in the organization, which should include the C-fact agendum, the current state of the organization, and a set of C-fact called reaction set which is the result of the execution of the action.

- I consists of a set of intentions, called the intention base.

- S consists of the state base, which contains the current state, or in other words, the set of facts and relationships that the organization may need to know in order to perform C-acts or P-act.

- P consists of a set of P-facta, called the production base.

The model of the elementary organization can be called a crispie, in which every elementary actor role is the kernel of a crispie. So, an organization can be considered a network of collaborating crispies, also called crispienet, for which a visual modeling technique is provided. Figure 11 depicts the constructs that can be used in the modeling of the organization.

An actor role can be elementary, if it cannot be decomposed; otherwise it is called a composite actor role. An actor role can initiate or execute a transaction type.
An elementary actor role can only execute one transaction type. A transaction type is composed of a production bank and a coordination bank and its graphical representation is the conjunction of both representations. The production bank contains P-facts and relationships pertaining to the transaction type that have been initiated. The coordination bank contains all C-facts and relationships created by either the initiator or by the executor of the transaction. An information link between the actor role and a transaction symbol means that the actor has access to the coordination bank or to the production bank. The basic construct used in a crispienet generally contains two actors, a initiator and a executor, and a transaction type to which the actors are connected through the initiator link and a executor link. Moreover, a transaction can be self-activated if a single actor is both initiator and executor.

A set of viewpoints (referred to as aspect models) capture the production of ontological facts and corresponding acts, forming a coherent whole: (i) Construction Model (CM), which specifies the identified transaction types and the associated actor roles; (ii) Process Model, which details for each transaction type defined in the CM, the corresponding transaction pattern; (iii) Action Model, which specifies the action rules that each actor has to follow in the corresponding transaction types; (iv) State Model, specifies the object classes, fact types, result types and co-existence rules that are contained in the Action Model. Figure 12 represents an Actor Transaction Diagram (ATD) of a Library organization, which is part of the CM of the organization. The gray rectangles represent composite actors (in other words, actors that can be decomposed in other actors); the circles represent transactions; and the connectors with a square represent the executor of the transaction, while the other type of connector represents the initiator. The Design and Engineering Methodology for Organizations (DEMO) provides a process for the development of the models, which will be described in more detail in section 5.3.4.

4.7 The Business Motivation Model

The Business Motivation Model (BMM) [65], proposed by the Open Management Group (OMG), provides a conceptual framework for the development, communication, and management of business plans, identifying motivational factors behind it, the elements required for its formulation, and relationships between the elements. Figure 13 depicts the main the relationships between the main concepts of the specification.

Four major concepts are defined: End, Mean, Influencer, and Assessment. An End is "what an enterprise wants to be". Ends can be the Vision for the organization (what the organization wants to be); or a Desired Result, which can be either a Goal (long-term, comprised of Objectives) or an Objective (short-term, component of Goals). A Mean is "what an enterprise has decided to do in order to become what it wants to be", in other words, to achieve its Ends. Means can be the Mission, which describes what an organization does in order to achieve the Vision; Courses of Action, which can be a Strategy or Tactic; and Directives, which govern the Courses of Action. An Influencer is "something that can cause changes that affect the en-
terprise in its employment of its Means or achievement of its Ends". An Influencer can be External (from outside the organization; i.e., Applicable Legislation) or Internal (from within the organization; i.e., Available Resources). An Assessment is "a judgment about the influence" of an Influencer "on the enterprise’s ability to employ its means or achieve its ends", with decisions stemming from that being reflected in changes to the Ends and/or Means.

4.8 Summary

The modeling frameworks described above belong to different areas and thus have a different aim: RM-ODP is appropriate for software systems; Archimate and TOGAF are aimed for information systems; DoDAF and MODAF are appropriate for military systems and military organizations; and EO and BMM are appropriate for organizations. Despite that, all those modeling frameworks offer a meta-model and, sometimes, a set of viewpoints that guide the development of models using the concepts defined in the meta-model.

Some of the meta-models described already provide the concept of capability, in particular the EA frameworks such as TOGAF, DoDAF and MODAF. Others do not, but some of the dimensions (section 3.4) that constitute a capability are explicitly modeled. For instance, in the case of RM-ODP, although it models time and distribution aspects at the computational level, it misses on those same aspects on the enterprise viewpoint. ArchiMate has an overall coverage of all dimensions, but misses on the Why dimension, a fact that is going to be addressed in the next version of the framework [64]. BMM focus on motivational aspects but misses on the When and Where dimensions, a fact that is acknowledged in its specification, where it is considered to be out of scope [65]. Finally, EO fully models the process aspects, but misses on the motivational and resource positioning aspects.

5 Methods for Engineering Systems

In this section, relevant systems engineering methods from the areas of software engineering, systems engineering, and enterprise architecture will be described and analyzed.

5.1 System Development Life Cycle Models

Several system development models exist that have been applied to different disciplines of engineering, due to their generic nature. These system development process models are also referred to as Systems Development Life Cycle (SDLC) models. SDLC models can be sequential, if they follow a predetermined sequence of steps or activities, or iterative, if several iterations are made with the purpose of revisiting the activities so that changes to the requirements are more easily dealt with. The most common SDLC models are described below.

5.1.1 The Waterfall model

The Waterfall model was one of the first SDLC models, firstly documented in 1956 [66]. In 1970, Royce described the Waterfall process for the development of large software systems (although he did not name it as such), which consisted of seven sequential and ordered stages: System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations [67]. Each stage would produce a specific document, documenting the outcomes of the stage. Figure 14 depicts the process.

In the same work, Royce also proposed an iterative variant of the same process as a better approach for projects [66]. Yet, the straightforward version was the one that became widely known [68].

Several variants of the waterfall model exist, all of which share similarities, in which the needs of the stakeholders are identified, analyzed, and synthesized into a solution design to be implemented, validated and maintained (e.g., in [53], Sage presents a system development method based on the waterfall process). The advantages of using such a life cycle model for the development of systems are listed below [69]:

- dictates the assessment of the needs of future users before design takes place;
- encourages design before building;
enables tracking of project progress through milestones;

- ideal for projects where the needs of the stakeholders are well known and defined, and the environment surrounding the system is fairly stable (in \[66\], it is argued that this type of model is ideal for developing software that provides back-end functionality);

The main disadvantage with the waterfall model is its rigidness in face to changes in the requirements in later stages of the life cycle, which in non-linear scenarios might raise problems \[70\]. Failure to meet the needs of management due to the focus on the development of single applications; unambitious systems design; instability; user dissatisfaction due to the waiting until the end of the development cycle to be able to experiment with the system; problems with excessive documentation; and application backlog due to possible changes in the final product by customer request are also listed as disadvantages \[71\].

5.1.2 The V-model

The V-model is considered to be a variation of the waterfall model \[66\]. It is composed of sequential stages, depicting a top-down and bottom-up implementation approach. The top-down part of the model goes from requirements gathering to the design and implementation of the components of the solution. A testing plan is created as a result of each of these stages. The bottom-up part deals with the testing of individual components, testing the integration of these components, and testing the overall system, using the test plans developed during the top-down part of the model.

The V-model addresses different perspectives of development stakeholders. In \[72\] the perspectives are defined as: the user perspective, which poses requirements and is interested in a final product meeting these requirements; the systems’ engineer perspective, which is concerned with the conceptual design of the system and with the integration and testing of the system according to the design; and the contractors’ perspective, which is concerned with the design and development of individual components and verification and testing of those components.

Most of the advantages pointed to the waterfall model are also applicable to the V-model. The V-model has the additional advantage of the early preparation of test plans, which increase the probabilities of success. Unfortunately, the V-model also shares the disadvantages of the waterfall model, such as the lack of agility in face of changes in the requirements in advanced stages of the life cycle, thus deeming the V-model inappropriate for complex projects \[66\].

5.1.3 The Spiral model

The Spiral model is a iterative development life cycle model. Systems’ development is done in cycles, typically each addressing the following four activities \[66\]: Determine objectives, Identify and resolve risks, Development and Test, and Plan the next iteration.

This life cycle model uses prototyping, which tries to take advantage of increased user participation on the development process, lowering the risks of rejection or disappointment. Mock-ups of the system can be developed iteratively in order to better correspond to the desires of the users, allowing the validation of requirements. Normally, prototypes might end up discarded or in some cases sent to production. A disadvantage of prototyping is the possibility of creating false expec-
tations, since normally a mock-up only mimics the desired functionalities. Since it requires many iterations, prototyping might add costs in budget and schedule to the development process.

Each cycle results in a prototype which is then validated and tested. The inclusion of risk management activities prior to the development of prototypes ensures that the costs of each cycle are kept to a minimum and that schedule is not overrun. At the end of the execution of each cycle, a review of all the artifacts produced is made to ensure that the development is aligned to the wishes of the customer. With the completion of each cycle, the cumulative cost increases and the product becomes closer to the desires of customers [72], as represented by the vertical and horizontal axis of Figure 16.

The Spiral model is suitable for large-scale projects and its advantages lie in its risk-oriented approach [66] [74], and in its adaptability to changes [74]. Disadvantages pointed to this model are that it requires expertise in risk assessment, since any neglected risk might cause trouble [74].

5.1.4 The Incremental model

The Incremental model is an iterative form of the waterfall process, in which the development of a system is made in a series of increments adding new functionalities. It can be seen as a series of waterfall, each one resulting in an operational system, as shown in Figure 17. Normally, the first increment is the core product. Each increment is validated by the users and as a result new functionalities are added or improvement of the existing ones is made. This life cycle model distinguishes itself from Prototyping models since each increment focuses in delivering an operational product that is basically a stripped down version of the final product.

This type of development life cycle model can be useful since it provides the customer with a functional system requiring less time and effort [74]. Stakeholders can be involved early in the development process, which reduces the risks [66].

5.1.5 The SIMILAR process model

The SIMILAR process model was created for the systems engineering domain as an abstract process based on the way humans solve problems in diverse areas [75]. SIMILAR stands for State the problem, Investigate alternatives, Model the system, Integrate, Launch the system, Assess performance, and Re-evaluate, which are also the names of the activities carried out through the iterative process. Figure 18 depicts the SIMILAR process.

The first activity aims the identification of the problem which the system will solve, with the inputs being the stakeholder’s needs and the list of constrains to the project, and the main output being the list of requirements that the product system should address. The Investigate alternatives activity has the objective of exploring and evaluating different alternatives for the product system, with the inputs being the requirements gathered in the previous activity and other architecture constrains, and the main output being a baseline architecture design. The Model the system activity involves the development of a detailed and comprehensive description of the system to-be implemented, with the inputs being the baseline architecture design and the main output being a model-based system design specification. The Integrate involves the implementation of the system using the specifications developed in the previous step and at the same ensure the correct integration with interfacing systems. The Launch of the
system activity involves the transfer of the system to the client, including validation and verification activities, to ensure that the system is functioning according to the needs of the client. The Assess performance activity deals with the measurement of the performance of the system so that any necessary improvements can be made. Finally, Re-evaluate is a continuously-run activity which deals with monitoring all the development process and its outputs and with basis on that revisit any previous activity or even decide for the disposal of the system.

5.2 Software Engineering Methods

The domain of software engineering has a strong history of methods for software systems development. In recent years, there has been a tendency to increase the agility of development methods, so agile methods are also described. Some of the methods described in this section are also surveyed in [31].

5.2.1 The Cleanroom method

Cleanroom is a method that originated in the 80s, with the following priorities: (i) to prevent any defects to happen rather than defect removal later in the system life cycle, through the use of human mathematical verification; and (ii) valid statistical certification of the quality of the software through user testing [77]. Its main purpose is to eliminate debugging from software engineering and provide high-quality software products through representative user testing. Figure 19 depicts the development process provided by Cleanroom, containing several sub-processes, which are numbered within brackets.

Cleanroom advocates a life cycle consisting of incremental software releases which add functionality to the product [76]. Processes (1-4) are management processes with the purpose of managing the development and include: project planning, project management, performance improvement, and engineering change.

Specification processes deal with the design specification of the system and include: architecture specification (5), which spans the whole development cycle; requirements analysis (6); function specification (7); usage specification (8); and increment planning (9). Development processes deal with the development of an increment in engineering terms (10-12) include: software reengineering; increment design; correctness and verification. Certification processes deal with the testing and certification of the product with user participation and include: usage modeling and test planning (13); and statistical testing and certification (14). The certified product is then delivered to the customer and can undergo another iteration, if new functionality is desired.

5.2.2 The Rational Unified Process

The Rational Unified Process (RUP) was developed by Rational Software in 1987, and later acquired by IBM in 2003. RUP combines six best-practices commonly used in the software industry for the development of software [78]: Iterative software development; Requirements management; Use of component-based architectures; Model-based development of software; Software quality assurance; Control and manage any changes to software.

A development process is provided with RUP, which is divided in four phases, each one concluding with a major milestone [79]: Inception, which deals with the establishment of the business case and with the delimitation of the scope of development project, and con-
includes with the life cycle objectives milestone; Elaboration, which deals with the analysis of the problem domain, establish architectural foundations, planning, and analyze and mitigate any risks in the project, concluding with the life cycle architecture milestone; Construction, which includes the development of all components and features of the product and consequent test, and includes the initial operational capability milestone; and Transition, where the developed product is delivered, including the product release milestone. At the end of each phase an assessment is made of the progress towards the objectives of the phase.

The development process comprises several disciplines, as displayed in Figure 20. Each discipline in turn comprises a set activities producing a valuable result, the designation of the workers performing the activities, and artifacts produced, modified or used. Nine core disciplines are provided with RUP, comprising six process disciplines such as Business Modeling, Requirements, Analysis & Design, Implementation, Test, and Development; and three supporting disciplines, including Project Management, Configuration and Change Management, and Environment. Those disciplines might be used in several phases of the development with variable levels of intensity. Moreover, each iteration might involve the use of several disciplines.

RUP attempts to deal with changes to the environment through the fact that each phase can be composed of several iterations (a complete development loop), each resulting in an internal or external release of a functional software product. The Environment discipline is concerned with the preparation of the development environment and not to the business environment itself.

5.2.3 The Enterprise Unified Process

The Enterprise Unified Process (EUP) is a variant of RUP, which tries to minor or solve some of its pointed weaknesses, such as no support for the whole life cycle of the system, no explicit support for the modeling and development of multiple enterprise systems, and difficulty to put the iterative nature of the process to practice [80]. EUP adds two additional phases to the development life cycle: the production phase, which covers the operation and maintenance of the system, and the retirement phase, which covers the dismantle-ment of the system. With these additional phases, the whole life cycle becomes covered by the process.

Additional disciplines are also added to the process and are organized in three different groups: development, support, and enterprise. The Business Modeling, Requirements, Analysis & Design, Implementation, Test, and Deployment are adopted from RUP and are part of the development group of disciplines. Also adopted from RUP are the Configuration & Change Management, Project Management, and Environment disciplines, which are part of the support group, along with a newly defined discipline Operations & Support, which has bigger intensity in the Production and Retirement phases. A completely new set of disciplines is part of the Enterprise disciplines group: Enterprise Business Modeling, Portfolio Management, Enterprise Architecture, Strategic Reuse, People Management, Enterprise Administration, and Software Process Improvement. These disciplines provide the organizational framework for the success of the development process. Several of the disciplines adopted from RUP were also updated in terms of life cycle coverage (e.g., the Test discipline starts earlier in the life cycle, more precisely on the Inception phase). Figure 21 depicts the development process.

Besides being iterative, the EUP attempts to deal with change through the inclusion of several enterprise management disciplines, most notably the enterprise business modeling discipline. This discipline has the
5.2.4 The ICONIX method

ICONIX intends to be an use case driven method with strong use of UML and a focus in the traceability of requirements. It is considered to be a lightweight method, since it uses only four UML diagram types of the twelve available. ICONIX is divided in four stages: Requirements resulting in the Requirements Review milestone; Analysis/Preliminary Design which results in the Preliminary Design Review; Detailed Design finishing with the milestone Critical Design Review, and Implementation ending in the milestone Delivery.

The Requirements stage deals with the assessment of the functional requirements and domain modeling, and carries on with the assessment of behavioral requirements through the use of a graphical user interface (GUI) prototype. On the Analysis/Design stage, a robustness analysis of the use cases is made, and the logical software functions needed to implement the use cases are determined. The Detailed Design stage deals with the elaboration of sequence diagrams detailing the implementation of the use cases and with the revision and clean up of the domain model, which at this point should be a UML class model. Finally, the Implementation stage deals with the coding and testing of the software units, integration of the software units implementing use cases, and with the revision of the code and models in order to prepare for a new iteration. Figure 22 depicts the process.

5.2.5 The Method for an Integrated Knowledge Environment 2.0

The Method for an Integrated Knowledge Environment (MIKE) 2.0 is an open-source method with the purpose of delivering information management software systems, through an iterative and continuously improved process. It is continuously built through experience and best practices sharing by a community that participates through a web 2.0 platform that follows the wiki model.

The development process is divided in five phases: business assessment and strategy definition blueprint, technology assessment and selection blueprint, information management roadmap and foundation activities, design increment, and incremental development, testing, deployment and improvement. Figure 23 depicts the development process.

The two first phases are mainly strategic. The first phase deals with raising awareness for information management concepts, establishing strategic business requirements, assessing the "as-is" situation and the "to-be", designing the conceptual architecture, and establishing a system delivery plan. As for the second phase, it completes the blueprint, concentrating on the technical aspects, such as the refinement of the requirements, the definition of the technology architecture, the determination of standards, policies, and baseline infrastructure to support development, and the development of an overall implementation and deployment plan.

The remaining phases correspond to the implementation. The third phase deals with the refining of the strategic business requirements into a technical specification, completing the solution architecture, establishing technical standards, defining the development and delivery environment, and planning for the current cycle of implementation. The fourth phase completes the design of the architecture developed on the course of the previous phase. Finally, the fifth phase is broken into four groups of activities: develop, which transforms the technical design into working modules; test, which tests the solution; deploy, which delivers the system into the production environment; and continuous improvement, which focuses on improvements to the existing functionality. Each of the described phases is then composed of several activities, each one resulting in deliverables. The activities can be further broken down in tasks.

Although generic, the development process described above aims to deliver solutions that occupy a broad spectrum of information management: business intelligence, information asset management, access, search and delivery, enterprise data management, enterprise
content management, information management strategy, architecture and governance, and composite solutions which combine several features of the other solutions. MIKE2.0 provides guidelines for the development of solutions belonging to those groups, describing how to apply the process to those specific cases. The solution development follows an architectural reference model denominated Strategic Architecture for the Federated Enterprise (SAFE), which depicts generic capabilities needed to implement most of the projects and capabilities that are specific to the area of Enterprise Information management. Governance guidelines and organizational models are also provided so that solutions are effectively delivered and managed throughout their life cycle.

5.2.6 The Dynamic Systems Development Method

The Dynamic Systems Development Method (DSDM) provides a framework for practicing Rapid Application Development (RAD). It is maintained by the DSDM Consortium.

RAD aims to fasten the development cycle of a software system, through the use of prototypes, rapid development tools, and close participation of the stakeholders in the development process. Requirements are gathered from the stakeholders and validated with the construction and test of prototypes, with the participation of stakeholders, in an iterative way. When a consensus is reached, the system follows into production. It makes use of the Computer-Aided Design and Engineering (CASE) tools for the development of data and process models, and to ensure the rapid development of prototypes. DSDM provides a framework of controls and guidance in order to do proper RAD.

DSDM aims to plan the amount of functionality needed according to the resources available. It consists of one pre-project phase, which has the aim of selecting projects and providing resources, a project phase, which is comprised of additional phases that will be described below, and a post-project phase, which focuses in maintaining the developed system operational.

The project phase consists of two sequential phases, feasibility and foundations, and of three iterative and incremental phases, exploration, engineering, and deployment (Figure 24).

The feasibility phase determines if the use of DSDM is adequate for the type of system to be developed, resulting in a feasibility report and on an outline plan for development. A fast prototype might be built during this phase if the requirements are not well known. In the foundations phase, the characteristics of business and technology are analyzed, in order to better know the domain. This phase sets the ground rules for later developments.

In the exploration phase, a partial solution is produced according to the refined requirements of the stakeholders, which is later completed in the engineering phase, which results in operational increment ready to be deployed. During the deployment phase, the users are trained, the system is deployed into the user environment, and the system is validated by the users. If problems are found during user validation, previous development phases might be revisited.

5.2.7 The eXtreme Programming method

The eXtreme Programming (XP) development method owes the word "“extreme”" due to the fact that it takes agile principles to the extreme. It is strongly based on the agile principles of involving the customer, short release cycles, respect for the workers, and simple and tested code, among others.
Figure 26: The ASD development process (Source: [31])

Its process consists of five phases (Figure 25): Exploration, Planning; Iterations to Release; Productionizing; Maintenance; and Death [32].

On the Exploration phase, story cards are written by the customers, each depicting a desired feature. The development team develops a prototype of the system. The Planning phase deals with prioritizing the implementation of the stories contained in the story cards. Commitments are made with the customers with respect to the functionalities to be implemented, effort to be spent, and schedule for the first release. In the Iterations to Release phase, several iterations are planned according to the defined schedule. The first iteration creates the architecture of the system, implementing the stories that are central to the desired functionality of the system. The stories to be implemented for each iteration are selected by the customer. The customer also tests the functions implemented at the end of each iteration. Pair programming is used during the iterations.

The Productionizing phase deals with additional testing done prior to the release of the system, where changes to the system might be decided upon and implemented, in shorter iterations. Changes not implemented by decision are documented for later implementation. In the Maintenance phase, the system is kept running and additional iterations are made to implement other scheduled features or interventions. New team members might be added due to the increased focus on maintenance and less focus on development. The Death phase signals the closing activities of the development process. If the system fully satisfies the customer, the final documentation is elaborated. If for any reasons the customer is not satisfied or if costs rise beyond bearable levels, the process is also finalized.

5.2.8 The Adaptive Software Development method

The Adaptive Software Development (ASD) method is an iterative and adaptive development process comprising three phases: Speculate, Collaborate, and Learn.

Figure 27: The FDD development process (Source: [34])

It assumes that development environments are highly volatile and as such instead of planning, one might just speculate on what the system should do. Then, intense teamwork should be used on the design and implementation, in order to cope with change. After that, a review of the results should be carried so that the team can learn from the experience.

Furthermore, the three phases are comprised of other (sub)phases. The Speculate phase comprises the Project Initiation phase, which sets the scope and preparation of the teams, followed by the Adaptive Cycle Planning phase, which sets the schedule and time frames for the project. The Concurrent Component Engineering phase occurs during the Collaborate phase, where several components of the system are developed concurrently. Finally, the Learn phase is comprised of the phases Quality Review, which deals with the group revision of the developed components, and Final Quality Assurance and Release, which consists on a final validation and deployment into the operational environment. Several iterations might occur to solve any problems encountered during Quality Review which results in revisiting the Adaptive Cycle Planning and Concurrent Component Engineering phases. Figure 26 depicts the development process.

5.2.9 The Feature Driven Development method

Feature Driven Development (FDD) is a development approach which focuses in the design and implementation of systems, leaving out of its scope other traditional life cycle phases [31]. It is built upon a set of nest practices such as domain object modeling, development by feature, individual class (code) ownership, feature-developing teams, quality inspections, regular builds, configuration management, and progress reporting [34]. Its development process consists of five subprocesses: Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature, and Build by Feature, the last two subprocesses being it-
The first phase deals with the elaboration of object models for capturing the problem domain from the scope, context, and requirements descriptions obtained using methods external to FDD. After that, the second phase deals with identifying the functions that the system should perform, culminating with the elaboration of a prioritized list of features that the system should possess. The third phase focuses on the planning and scheduling of the development of each feature. Finally, the last two subprocesses deal respectively with the design of how each feature should be realized through the use of sequence diagrams, and with the coding and testing of each feature. The features can be designed and built iteratively and even concurrently if the work is split through the development team. Figure 27 depicts the process.

5.2.10 The Scrum method

Scrum is an iterative and incremental development method where each iteration occurs one after another, each delivering a functional increment of the product, according to a list of requirements [87].

The project starts with the vision of the system to be developed, including estimated Return on Investment (ROI). From that vision, a plan emerges which contains a product backlog, which is a list of functional and non-functional requirements. Then, the product backlog is prioritized, so that base and value-generating functions are developed first. The development of the system is then made in iterations, also called sprints, which should take about 30 days. Each sprint is planned and negotiated taking into account the backlog priority and the resources of the development team, resulting on a selection of product backlog to be implemented during a sprint, and on a sprint backlog containing the tasks the development team has to complete in order to deliver the agreed functionality. During the sprint, a daily inspection occurs with individual team members inspect each other’s activities, making corrections when needed and synchronizing the work of each team member with everyone else.

At the end of each sprint, the results of the iteration are demonstrated to the stakeholders and planning is made concerning the remaining sprints. A retrospective meeting is also hold in order to revise the development process to make it more enjoyable and effective for the development team. Figure 28 depicts the Scrum development process.

5.3 Enterprise Architecture Methods

The management of the architectural artifacts belonging to an EA is seen as a major challenge to the research field of EA management, which has the aim of aligning business and IT and optimizing their interaction [88, 89]. Several enterprise architecture methods for the development and management of EA exist, including methods that are aimed for areas that are usually associated with traditional systems engineering, such as the DoDAF 6-Step Architecture Development Process. We describe some of the most common and relevant approaches below.
5.3.1 The TOGAF Architecture Development Method

The TOGAF Architecture Development Method (ADM) is a method for the development and guidance of the development of an architecture practice that is provided with TOGAF [22]. It can be considered the core of TOGAF.

The ADM as the heart of TOGAF consists of a cyclical process divided in nine phases, as shown in Figure 29. After a preliminary phase in which the context, relevant guidelines and standards and the goals of the architecture process are identified, the main process begins with the elaboration of an architecture vision and the principles that should guide the architecture. This provides the basis for subsequent phases: business architecture, information systems architecture, and technology architecture. On this basis, solutions are developed (opportunities and solutions phase), and migration and implementation are planned and governed (migration planning and implementation governance phases). Finally, architecture change management ensures that the architecture continues to be fit for purpose. The ADM can be adapted for various purposes, and in more complex situations, the architecture can be scoped and partitioned so that several architectures can be developed and later integrated using an instance of the ADM to develop each one of them.

5.3.2 The DoDAF 6-Step Architecture Development Process

DoDAF claims to be a method-based approach to architecture, providing a six-step process for the development of architecture descriptions and a series of non-prescriptive viewpoints which can be used on the architecture development process [20].

The first step consists in determining the intended use of the architecture, dealing with the definition of the purpose of the architecture, the methods to be used in the development, the categories of data needed in order to carry out the process, the impact of the architecture work, and the determination of a process for measuring success. The second step deals with the determination of the scope of the architecture, which establishes the boundaries for the architecture work. The third step consists on determining which data is required for supporting the architecture development, which deals with the assessment and identification of the data entities, attributes, and relationships required for the architecture work. On the fourth step, architectural data is collected, organized, correlated, and stored in an architectural tool, to be later used in the definition of views. The fifth step deals with the verification of the conformance of architectural data to the requirements and objectives settled during previous steps. If any problems are encountered during this step, a new iteration starting in step three should be performed. Finally, the sixth step deals with the elaboration of views for presentation and decision-making purposes.

The viewpoints developed during the application of the method are composed of models and supported by the DoDAF Metamodel, which guides the architecture content and formally defines the vocabulary for architecture development. In particular, the capability viewpoint supports incremental acquisition and visualization of evolving capabilities to support procurement in complex situations. Figure 30 depicts the architecture development process.

The architecture development process is a single phase of the overall life cycle, which comprises the use
5.3.3 The Federal Enterprise Architecture Framework method

The United States Federal Enterprise Architecture Framework (FEAF) program aims to "build a comprehensive business-driven blueprint of the entire Federal Government" [92]. It consists of a series of reference models which provide a common terminology to be used by the different federal agencies, facilitating gap and opportunity assessment, and cross-agency collaboration [92]: Performance Reference Model (PRM), for performance measurement; Business Reference Model (BRM), for business areas; Service Component Reference Model (SRM), for service components supporting business and performance objectives; Technical Reference Model (TRM), for technical components and standards supporting the services; and Data Reference Model (DRM), for data and data management processes.

FEAF views an enterprise as being comprised of segments, which are individual elements of the enterprise which describe core mission areas, and common or shared business or enterprise services. Core mission area segments are specific to the mission of agencies (e.g., health), Business Services are foundational and used by all the agencies (e.g., human resources), and Enterprise Services are crosscutting and span diverse segments (e.g., security management). Figure 31 depicts segments and services.

In order to develop a results-driven architecture, a performance improvement life cycle is provided, comprising three phases [91]: Architect, Invest and Implement. Each of the phases can be broken down into processes, as shown in Figure 32. In the Architect phase, EA is developed representing a shared view of the as-is and the to-be of the enterprise, segments are identified and prioritized, and segment architecture is developed, transitioning into the next phase. On the Invest phase, the implementation and funding strategy for solutions described on the segment architecture is defined, and a program management plan is created to implement the solutions, transitioning to the next phase. Finally, on the Implement section, the solutions are implemented according to the program, and the performance is measured on the solutions, after those are deployed, in order to check if the desired results are being achieved. Feedback from the measurements is inputed into the enterprise and segment architecture development processes.

5.3.4 The Design and Engineering Methodology for Organizations

The Design and Engineering Methodology for Organizations (DEMO) provides a process for developing the ontological models of an organization [42]. It has the following steps: (i) Performa-Informa-Forma Analysis, which from the analysis of the business scenario of the organization, divides the knowledge present on the scenario description in ontological, datalogical, or ontological; (ii) Coordination-Actors-Production Analysis, which divides the ontological pieces found in the previous step in C-acts/results, P-acts/results, and actor roles; (iii) Transaction Pattern Synthesis, which according to the results of the previous step, associates transaction types with result types; (iv) Result Structure Analysis, which analyzes the structure of related transaction results in order to detect dependency relations; (v) Construction Synthesis, which deals with the conception of the ATD of the organization, depicting the initiator and executor actor roles involved in a transaction; and (vi) Organization Synthesis, which deals with choosing the section of the construction model which will be the focus of the modeling. This last step concludes with the elaboration of the ATD. From this on, the other aspect models can be easily derived through the usage of transaction patterns.
The Generic Systems Development Method (GSDM) constitutes a way of using the ontological models to develop information systems [93]. Through the use of DEMO, the function design of the organization can be reverse engineered. Functional requirements can then be derived from the models, and along with functional architecture principles, which put restrictions unto the design, can be used to derive the black box model of the system. That model along with constructional requirements and architecture principles can be used to engineer the construction of the system, which might use any information systems ontology in its design. Figure 33 depicts the GSDM.

5.4 Systems Engineering Methods

Systems engineering is a domain with a broader scope than that of software engineering, with the aim of engineering complex systems combining the logical dimension of systems with the physical dimension. Several methods exist of which we describe some of the most prevailing below. Some of the methods below were already surveyed in [94].

5.4.1 The Vitech MBSE Method

The Vitech Corporation is a company specialized in systems engineering and architecture solutions. One of the solutions provided by Vitech is a Systems Engineering method which adopts an incremental and layered development process, also known as the "Onion Model" [95]. Apart from the typical requirements analysis and design, the method includes placeholders for specialty engineering, with assessment of the implications of the requirements brought by specialized areas into the overall architecture design.

Figure 34 depicts the development process. The method includes a context definition process, requirement analysis, functional analysis, and architecture analysis. Static and Dynamic Verification and Validation also takes place. Concurrently, for each specialty, specialty requirements analysis, specialty functional analysis, and specialty architecture analysis is made, which might result on the review of the overall requirements, functions, and architecture, and design of specialty functions and architecture. The design of the overall solution is then reviewed at the end of the process.

As referred above, the development process is incremental, with each iteration further detailing the architecture design. With each successful iteration, a layer is "peeled" and the next layer of detail design begins. All concurrent engineering activities pertaining to a layer should be completed before moving to the next layer. The iterations end when the level of design satisfies the parts involved. This allows architecture designs to be reviewed early in the process, preventing possible risks. In case of problems, the layer above can be revisited. Revisiting layers two or more levels above can represent a major problem in terms of cost and schedule. Figure 35 depicts the onion model.

5.4.2 The Rational Unified Process for Systems Engineering

The Rational Unified Process for Systems Engineering (RUP-SE) is a derivative of the Rational Unified Process, already described in section [5.2.2] [96]. The process still adopts UML as the modeling language as
5.4.3 The Object-Oriented Systems Engineering Method

The Object-oriented Systems Engineering Method (OOSEM) is a model-based top-down (functional decomposition) approach to systems engineering which combines Object-oriented concepts with traditional systems engineering foundations [56].

OOSEM adopts a recursive development process based on the V-model, which includes the following activities: the analysis of stakeholder needs; the definition of system requirements; the definition of the logical architecture, with decomposition and allocation of system functionality into logical components; the synthesis of the allocated architectures, with the allocation of the logical components into software, hardware, data, and manual procedure components; the optimization and evaluation of alternatives, which is carried through all the other activities in order to optimize the architecture and to aid on decision-making; and the validation and verification of the system, to check from time to time, during the development process, if the system is fulfilling stakeholders’ needs. Both UML and SysML modeling languages can be used to support the specification, analysis, design and verification of systems (both hardware and software). Figure 37 depicts the described activities and resulting artifacts.

5.4.4 The Rational Harmony method

The IBM Rational Harmony is an iterative model-based top-down approach to the engineering of systems, which uses both SysML and UML as the modeling languages [98]. Its emphasis resides in the identification and validation of the functionalities and behavior of the system, in abstract, high-level terms. The functionalities are then allocated to sub-system components, at a lower-level of abstraction, in a process which resembles a v-model life cycle (Figure 38). The requirements analysis and system analysis & architecture are part of the systems engineering work, which has the purpose of allocating system functionality to the physical architecture. The artifacts resulting...
from those activities are then handed off to sub-teams that are responsible for developing subsystem specifications from the inputed artifacts. All the activities following the hand-off are part of the *software engineering* work. All the models and requirements are contained in a central repository. Those specifications are developed and implemented through a spiral-like analysis-design-implement-test cycle. Then, functionalities are developed incrementally, with each increment adding additional functionality to the overall system.

### 5.4.5 The Object-Process Methodology

The Object-Process Methodology (OPM) combines formal visual models with constrained natural language sentences with the aim of modeling the structure, behavior, and functions of the system in an integrated way, using a single model for that purpose [94]. Three types of entities are defined within OPM: Object, which is a thing that exists physically or mentally; Process, which is a pattern of transformation that an object undergoes; and State, which corresponds to the state of an object. A process can create, consume, or alter an object, affecting its state.

In the visual representation, known as Object-Process Diagrams (OPDs), objects adopt a rectangle shape, processes adopt an ellipse shape, and states adopt a rounded rectangle shape. Links between the elements are also a part of OPDs, such as structural links, which describe static relations (i.e., aggregation, generalization, characterization, instantiation), and procedural links, which connect entities to describe the behavior of the system. The textual representation is known as Object-Process Language (OPL), and represents the textual counterpart of OPDs.

In [99], the authors of OPM describe a system development process through reflecting meta-modeling, or in other words, the method is modeled by the means and tools that the method itself provides. Figure 39 depicts the top-level development process metamodel, which uses both OPD and OPL. Through the use of refinement/abstraction mechanisms present in OPM, it is possible to zoom into the System Developing element, which allows for a more detailed view of the development process. Figure 40 depicts the details of the System Developing process. Invocation links are present in all the sub-processes so that the System Developing process can be restarted (i.e., a new iteration) in case a problem is found. Further zoom in operations are possible on the sub-processes present in System Developing, but are not to be described in here.

### 5.4.6 The State Analysis method

The State Analysis method was created by the NASA Jet Propulsion Lab (JPL) specifically to represent dynamic behavior, as a complement to the more traditional functional analysis [100]. The purpose is to model complex dynamic interactions specially when developing unmanned systems which cannot be captured using functional analysis, and to better translate the requirements from systems engineers to software engineers by providing a common language. According to [101], state analysis allows for: (i) discovering, characterizing, representing, and documenting the states of a system; (ii) modeling the behavior of states and relationships among them, including information about hardware interfaces and operation; (iii) capturing the mission objectives in detailed scenarios motivated by operator intent; (iv) keeping track of system constraints and operating rules; and (v) describing the methods by which objectives will be achieved.

For that, a state-based control architecture framework is provided where: the state becomes explicit; state estimation is decoupled from state control; the interface between the control system and the system under control is provided by hardware adapters; models are used for execution and for higher level planning;
the operations adopt a goal-directed closed loop; and 
the straightforward mapping into software is possible.

Figure 41 depicts the state-based control architecture.

State analysis allows for (i) Modeling behavior in terms of system state variables and relationships between them (state-based behavioral modeling); (ii) Capturing mission objectives in detailed scenarios motivated by operator intent (goal-directed operations engineering); and (iii) Describing the methods by which objectives will be achieved (state-based software engineering). A process for performing state modeling is provided in the method, which goes through the identification of needs and objectives, identification of the state models for the identified state variables, identification of measurements for the estimation of state variables, definition of measurement models for the identified measures, identification of commands for controlling the state variables, and definition of command models for the identified commands. This process is iterative and is executed until every state variable is accounted for or, if new supporting objectives are discovered during the design of the control system, until the scope of the mission has been covered. By using a state database for storing the elaborated models, documentation can be generated in a way that it can be used to better inform software engineers.

Although it represents a different paradigm, state analysis can complement functional analysis. Figure 42a represents functional analysis elements and relationships, while Figure 42b represents both functional and state analysis elements related, bringing together the benefits of the two approaches.

5.4.7 The Soft Systems Methodology

The Soft Systems Methodology (SSM) was developed primarily by Peter Checkland to face so called "soft problems", such as the ones present in organizational context, although it is argued that it does not distinguishes from the nature of the problems, as long as these are situated in a real life context. It differs significantly from traditional systems engineering, which focus in well defined systems and problems with predominant technical factors (also called "hard systems"), focusing instead in finding solutions to ill-defined, complex problems, with the human or social factor involved.

The method involves four activities that are continuously performed in a learning cycle in order to deal with a problematic situation: Finding out about a problematic situation, which comprises the elaboration of a representation of a situation as a picture, and also user, social, and political analysis of the situation; Building purposeful activity models, which from the analysis done in the previous step leads to the development of activity models, each based on a worldview, depicting a series of activities that can be used to tackle the problem; Exploring the situation, which involves the discussion of the models developed in the previous step; and Defining action to improve the situation, which involves reaching agreements with all the involved stakeholders with respect to actions to be determined to resolve the problematic situation. Figure 43 depicts a purposeful activity model elaborated in to deal with the problematic situation of managing internally generated information in a law firm, which presented several issues, for instance, dispersed and duplicated information which made knowledge sharing difficult.

5.5 Summary

Despite belonging to different research domains, the methodologies presented in the previous sections as well as the techniques for their engineering share common concepts. Furthermore, the complexity of the sys-
tems existing in the contemporary world and their interdisciplinary nature makes impossible the dissociation of these domains and corresponding practices.

Table 8 summarizes the software engineering methods described above in terms of the life cycle model that they follow, their suitability, the keywords that can be used to characterize them, and of the guidance that is given for their tailoring for specific projects or situations. From the observation of the table, it is possible to verify that none of the described software engineering methods follows a sequential life cycle model. All the methods are iterative, with some also being incremental (itself a form of iterative). The agile methods, i.e., DSDM, XP, ASD, DD, and Scrum, are noticeable for being suited for environments with unstable requirements, but varied in terms of the scale of the projects that their suited for. All the methods show different strengths, which can be verified by the keywords that can be used to characterize them. Cleanroom has obviously the main objective of preventing defects from happening instead of solving them after they happen. RUP is known for being an use case driven and iterative approach to software development that employs different disciplines for managing the life cycle of software systems. As for EUP, despite being a derivative of RUP, it distances itself from it by having a more holistic coverage of the life cycle of a software system, providing support for the maintenance and retirement of the system. ICONIX is also use case driven, but is far more minimalistic than RUP. MIKE 2.0 is noticed for being an open source methodology that is the fruit of a collaborative effort. Agile methods are known for being flexible and adaptable, but their strengths are quite different since they offer different approaches. DSDM follows the Rapid Application Development paradigm. XP is known by its lightweight and simplicity, as well as for its emphasis on testing. ASD uses creative prototyping for capturing emergent requirements. FDD is a feature-driven development approach. Finally, Scrum has its emphasis in the flexibility, adaptability and productivity through planning of the development.

Table 9 summarizes the enterprise architecture methods described above in terms of the life cycle model that they follow, their suitability, the keywords that can be used to characterize them, and of the guidance that is given for their tailoring for specific projects or situations. Almost all the methods are iterative, with the exception of DEMO. TOGAF, FEAF, and DEMO are suited for the design and engineering of enterprises, while DoDAF is targeted for US military organizations. All the methods differ in approach, with the most noticeable difference being that of DEMO, which represents a paradigm shift from the traditional enterprise architecture approaches, focusing on the commitments existing between actors and the coordination and production acts which result in coordination and production facts. The result is an ontological, implementation-independent model of the organization. On the other hand, the other methods have a strong focus in implementation issues: TOGAF cares about the development of an architecture vision aligned with the organization goals and strategy, and on the effective governance and change management of the resulting architectures; DoDAF is more inclined towards supporting information sharing that enables aligned and integrated decision making for the delivery of military capabilities; and FEAF aims for the development of reference models providing common terminology for the different segments of the organization to be used in the architecture development life cycle.

Table 10 summarizes the systems engineering methods described above in terms of the life cycle model that they follow, their suitability, the keywords that can be used to characterize them, and of the guidance that is given for their tailoring for specific projects or situations. The use of the V-model type of life cycle is noticeable in the systems engineering area, most notably in OOSEM and Rational Harmony. All the other methods employ an iterative life cycle management process. As for suitability, OOSEM, Rational Harmony, and RUP-SE are geared towards systems where software assumes an important place. Both Vitech MBSE and OPM authors argue that their application to the development of all kinds of systems should be successful,
Table 8: Summary of the software engineering methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Life cycle model</th>
<th>Suitability</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleanroom</td>
<td>Incremental</td>
<td>General</td>
<td>Defect prevention</td>
</tr>
<tr>
<td>RUP</td>
<td>Iterative and Incremental</td>
<td>General</td>
<td>Use case driven; Disciplines</td>
</tr>
<tr>
<td>EUP</td>
<td>Iterative and Incremental</td>
<td>Enterprise systems</td>
<td>Holistic; Retirement; Operations and Support</td>
</tr>
<tr>
<td>ICONIX</td>
<td>Iterative and Incremental</td>
<td>Systems with human interface</td>
<td>Use case driven; Minimalistic</td>
</tr>
<tr>
<td>MIKE2.0</td>
<td>Iterative</td>
<td>Information management systems</td>
<td>Open-source; Collaborative</td>
</tr>
<tr>
<td>DSDM</td>
<td>Iterative and Incremental</td>
<td>Environments with unstable requirements</td>
<td>Rapid application development</td>
</tr>
<tr>
<td>XP</td>
<td>Iterative</td>
<td>Environments with unstable requirements; Small to medium project teams</td>
<td>Lightweight; Simplicity; Testing</td>
</tr>
<tr>
<td>ASD</td>
<td>Iterative and Incremental</td>
<td>Environments with unstable requirements; Complex, large systems</td>
<td>Prototyping; Emergence; Creativity</td>
</tr>
<tr>
<td>FDD</td>
<td>Iterative</td>
<td>Environments with unstable requirements; Complex, large systems</td>
<td>Features; Iteration; Planning</td>
</tr>
<tr>
<td>Scrum</td>
<td>Iterative and Incremental</td>
<td>Environments with unstable requirements; Small to medium project teams</td>
<td>Flexibility; Adaptability; Productivity</td>
</tr>
</tbody>
</table>

while State Analysis is suited to the development of unmanned systems, although with human control at distance. Both OOSEM and Rational Harmony employ a top-down approach and make use of object-oriented modeling techniques, which is something that is shared with RUP-SE. Vitech MBSE adopts a layered approach to the development, where specialty engineers are responsible for the development of components. The most specific case is that of State Analysis which aims for the modeling of the dynamic aspects of the particular case of unmanned systems, focusing on the control of its functions according to the analysis of its state. As for OPM, it uses a mix of modeling elements with textual constraints to model any kind of systems. Finally, SSM claims to be a cyclical process aimed at solving problems with a strong soft component, using a systematic approach.

Using the life cycle representation suggested in [10], the methods described throughout this section were assessed for their coverage of the life cycle. Table [11] demonstrates that the majority of the software methods do not address the whole life cycle, with the exception of EUP. The Concept, Development, and Production phases are the ones that are normally addressed by methods, the exception being FDD. As for EA methods, both TOGAF and FEA present almost full coverage of the life cycle. Despite this, Retirement is not addressed in these methods. In turn, DoDAF seems to focus solely on the architecture development. DEMO itself only deals with the conception phase, leaving out the other phases. Similarly to software engineering, the majority of systems engineering methods cover the three first phases, the exception being OPM. This could be a sign of other important systems engineering activities and processes not being integrated into the development methods, functioning in separate.

A fact in common in the three domains is that the majority of methods do not cover the whole life cycle of the systems they focus in. The Concept, Development, and Production are traditionally addressed, but few methods address the Utilization, Support, and Retirement phases, which might pose a problem in complex and ever-changing socio-technical contexts, where stakeholder requirements often changed during the life cycle of a system. The issue becomes more serious if we take into account the purpose and holistic scope of software and systems engineering and enterprise architecture, as defined by their governing bodies, as can
be attested by the definitions given in the Introduction to this work. It can be argued that support for those phases is given in separate from the system development process by other processes (such as the ones presented in Section 2.4), but the lack of integration of those processes in an overall life-cycle management methodological framework might have consequences in the development, operation, and maintenance of a system. The EA field seems to be aware of those issues, since methodologies such as TOGAF provide a wide coverage of the life cycle and address several aspects traditionally addressed during the first phases of the life cycle throughout the whole life cycle (e.g., requirements management, risk management).

6 Method Engineering

The area of Method Engineering is defined as the "engineering discipline to design, construct and adapt methods, techniques, and tools for systems development". In this section, several approaches to the engineering of methods are presented: Process Meta-models and Situational Method Engineering. The related area of Enterprise Architecture Management is also discussed.

6.1 Process Meta-models

Process meta-models allow the specification of concepts and relationships for defining methods. Through the use of meta-models it is possible to formally describe, modify, or extend existing methods, or even create new methods, may it be for general purposes or for specific situations. In this subsection we describe three relevant process meta-models: SPEM, OPF, and SMDM.

6.1.1 The Software & Systems Process Engineering Meta-Model

The Software & Systems Process Engineering Meta-Model (SPEM) is a metamodel developed by the Object Management Group (OMG) based on Meta-
Table 11: Life cycle coverage of the methods

<table>
<thead>
<tr>
<th>Concept</th>
<th>Concept</th>
<th>Development</th>
<th>Production</th>
<th>Utilization</th>
<th>Support</th>
<th>Retirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleanroom</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUP</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUP</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ICONIX</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIKE2.0</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSDM</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASD</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDD</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scrum</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOGAF</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DoDAF</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEAF</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DEMO</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vitech MBSE</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUP-SE</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OOSEM</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rational Harmony</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OPM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>State Analysis</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSM</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Object Facility (MOF) meta-model and on the UML 2.0 meta-model, currently on its 2.0 version. Its purpose is to provide constructs that allow the definition of systems and software development processes and constituent components, leaving out specific constructs of particular areas (e.g., project management). It does not assign any particular modeling language for modeling process behavior nor does it provide a specific method for creating development processes. SPEM has the particularity of separating the content aspects of a method from the process definition. That way, it allows the definition of the content libraries which can be used in a determined process, according to the situation or context at hand.

The SPEM metamodel is divided in seven packages: Core, ProcessStructure, ProcessBehavior, ManagedContent, MethodContent, ProcessWithMethods, and MethodPlugin. The Core metamodel package contains classes and abstractions that provide the basis for classes in the other packages, most notably the WorkDefinition class. The ProcessStructure package defines the basis for static process models, supporting the creation of simple and flexible process models, defining classes such as Activity, Milestone, WorkProductUse, and RoleUse. The ProcessBehavior allows the extension of structural process models with behavior models. The ManagedContent package contains concepts for managing the textual description of processes, providing classes such as Guidance or Metric. The MethodContent package adds concepts for defining life cycle and method content elements, with the definition of classes such as TaskDefinition, RoleDefinition, and WorkProductDefinition. The ProcessWithMethods defines concepts for integrating the elements from the ProcessStructure and MethodContent. Finally, MethodPlugin defines concepts for the design and management of libraries of method contents and processes, such as MethodPlugin and MethodLibrary. Figure 44 depicts the relationships existing between the packages.

The SPEM specification provides an instantiation of the MethodPlugin classes named BasePlugin, which provides specializations of the meta-model concepts for the domain of software engineering. Those include ActivityKinds for defining life cycle models using commonly used terminology (e.g., Process, Phase, Iteration), CategoryKinds for defining groupings for categories of method contents (e.g., Discipline, RoleSet, Domain, ToolCategory), GuidanceKinds for providing terminology for commonly used guidance types (e.g., Guideline, Practice, Report, etc.), WorkProductKinds for common outcomes (e.g., Artifact, Deliverable, Outcome), and WorkProductRelationshipKinds for defining relationships among work products (e.g., Aggregation, Composition, ImpactedBy).
6.1.2 The OPEN Process Framework

The OPEN Process Framework (OPF) is an approach for designing system life cycle development processes. It is composed of three major parts: Metamodel, Method components repository, and construction and usage guidelines.

As shown in Figure 45, the core component classes of the metamodel are: Work Product, which models artifacts produced by the "collaboration of one or more producers during the performance of one or more work units" (e.g., Application, Architecture, System, Process, Diagram, etc.); Work Unit, which models operations performed by producers (e.g., Activity, Task, Work Flow, and Technique); Producer, which models entities producing, directly or indirectly, work products (e.g., Organization, Team, Role, Tools, and Persons); Language, which models the languages used on the documentation of work products (e.g., UML); endeavor, which models initiatives comprising the development or maintenance of one or more applications (e.g., Project, Program, or Enterprise); and Stage, which models "formally identified" time periods that provide organization to the work units (e.g., Cycle, Phase, Build, Milestone, Inch Pebble).

The method components repository contains component instances that can be selected and combined in order to create a method tailored to a specific situation. OPF suggests an incremental and iterative set of steps for producing methods: (i) Resource Management, which deals with the staffing of process team; (ii) Method Needs Assessment, which deals with the assessment of the specific process needs; (iii) Method Construction, which deals with the selection of appropriate reusable process component classes from the method components repository, tailoring of the selected process component classes according to the needs, and integrating the process component classes into the resulting method; (iv) Method Documentation, which deals with the elaboration of the documentation; (v) Training Delivery, which deals with the training of the staff and customer organization on the aspects of the developed process; (vi) Method Mandating, which deals with the formalization of the mandate for use of the method, by the management team; (vii) Method Consulting, which deals with the support given by the process team to the team that is going to use the process (the endeavor team); (viii) Quality Assurance, which deals with the evaluation of the effectiveness and efficiency of the process; and (ix) Process Framework Maintenance, which deals with the continuous iteration of the organizational process framework in order to keep the processes aligned with the needs of users and with the objectives of the organization.

6.1.3 The Standard Meta-model for Software Development Methodologies

The Standard Meta-model for Software Development Methodologies (SMSDM) provides concepts not only for designing a development process, but also for the modeling aspects of methods. In that sense, SMSDM provides the bridge between the two aspects of methods, going a step further than the other metamodels in terms of process definition. It has given origin to ISO/IEC 24744 - Software Engineering - Meta-model for Development Methodologies (SEMMD) [38].

There are three layers or levels of abstraction in SMSDM: meta-model, methodology, and process. The process level is constrained by the methodology level, and the methodology level is constrained by the metamodel. Classes in the metamodel are used in the cre-
Figure 46: High-level view of the SMSDM model (Based on: [108])

<table>
<thead>
<tr>
<th><strong>SMSDM</strong></th>
<th><strong>SPM 2.0</strong></th>
<th><strong>OPF</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Kind</td>
<td>WorkDefinitionParameter</td>
<td>WorkPerformance</td>
</tr>
<tr>
<td>Activity Kind</td>
<td>Kind (Instance)</td>
<td>Activity</td>
</tr>
<tr>
<td>Build Kind</td>
<td>Kind (Instance)</td>
<td>Build</td>
</tr>
<tr>
<td>Conglomerate</td>
<td>Category</td>
<td>Package, ProcessComponent</td>
</tr>
<tr>
<td>DocumentKind</td>
<td>Kind (Instance)</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>ContentDescription</td>
<td></td>
</tr>
<tr>
<td>Guideline</td>
<td>Guidance</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td>Language</td>
</tr>
<tr>
<td>LifeCycleKind</td>
<td>ProcessKind</td>
<td>Cycle</td>
</tr>
<tr>
<td>MilestoneKind</td>
<td>Milestone</td>
<td>Milestone</td>
</tr>
<tr>
<td>Outcome</td>
<td>Outcome</td>
<td></td>
</tr>
<tr>
<td>PhaseKind</td>
<td>Kind (instance)</td>
<td>Phase</td>
</tr>
<tr>
<td>PostCondition</td>
<td>Postcondition</td>
<td></td>
</tr>
<tr>
<td>Precondition</td>
<td>Precondition</td>
<td></td>
</tr>
<tr>
<td>ProcessKind</td>
<td>ProcessKind</td>
<td></td>
</tr>
<tr>
<td>ProducerKind</td>
<td>ProcessPerformer</td>
<td>Producer</td>
</tr>
<tr>
<td>Reference</td>
<td>Guidance</td>
<td></td>
</tr>
<tr>
<td>RoleKind</td>
<td>RoleDefinition, RoleUse</td>
<td>DirectProducer</td>
</tr>
<tr>
<td>Source</td>
<td>Guidance</td>
<td></td>
</tr>
<tr>
<td>StageKind</td>
<td>WorkDefinition, Activity</td>
<td>Stage</td>
</tr>
<tr>
<td>TaskKind</td>
<td>TaskDefinition</td>
<td>Task</td>
</tr>
<tr>
<td>TeamKind</td>
<td>Kind (instance)</td>
<td>IndirectProducer</td>
</tr>
<tr>
<td>TechniqueKind</td>
<td>Kind (instance)</td>
<td>Technique</td>
</tr>
<tr>
<td>ToolKind</td>
<td>Kind (instance)</td>
<td>DirectProducer</td>
</tr>
<tr>
<td>WorkProductKind</td>
<td>Kind (instance)</td>
<td>WorkProduct</td>
</tr>
<tr>
<td>WorkUnitKind</td>
<td>WorkDefinition</td>
<td>WorkUnit, WorkPerformance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ation of a method through instantiation on the methodology level. SMSDM adds the process layer in order to prevent the use of the concepts on the methodology layer as classes in the project layer, which is conceptually incorrect. By adding this additional layer, the flexibility on the definition of projects is increased. The relationship between corresponding elements of different layers is possible through the use of the powertype pattern, which allows the definition of element/kind pairs, in which one represents kinds of the other element (e.g., Document is a concept pertaining to the project layer and DocumentType pertains to the methodology layer).

Figure 46 depicts the core classes of the metamodel. The MethodologyElement class represents elements in the methodology layer, whereas the ProjectElement class represents elements in the project layer. MethodologyElement is specialized in Resource, which are elements used 'as is' in the project level, and Template, which can be instantiated. Template can be specialized in several other classes: StageKind, ActionKind, ModelUnitKind, ProducerKind, WorkUnitKind, WorkProductKind, and ModelUnitUsageKind. As for the ProjectElement class, it can be specialized in Stage, Action, ModelUnit, Producer, WorkUnit, WorkProduct, and ModelUnitUsage. Finally, Resource can be specialized in Language, Constraint, Guideline, Notation, and Outcome. Further specializations of these classes and relationships exist but are not shown here for space constraints.

6.1.4 Equivalence between the meta-models

The three meta-model specifications described above present different approaches on the modeling of methods. One way of comparing these approaches is map between the terminology used in the different specifications. Table 12 presents a revised version of the mappings described in [108], which used SPEM 1.1.

From the analysis of the table it is possible to see that both SMSDM and OPF have an emphasis on the modeling part of methods, whereas SPEM 2.0 is not present in SPEM. It is also possible to see that SMSDM and SPEM 2.0 are somewhat aligned, since the majority of concepts can be mapped to each other. This was not possible in SPEM 1.1, since it was lacking the Kind element provided in SPEM 2.0.

6.2 Situational Method Engineering

Situational Method Engineering (SME) aims to develop customized methods for a specific organization or project. It can be considered the solution for the problem of the selection of the most appropriate method for an organization or project [104]. SME has the aim of studying: how to create method fragments; how to store and retrieve method fragments; how to assemble a complete method from the fragments; and how to formalize the fragments. In SME, atomic components of methods are stored in a repository, also denominated method base, along with information on its characteristics or context of usage, so that they can be chosen and combined into a method according to the specific context of application.

Different concepts and definitions exist for the atomic components of methods, including method fragments, method chunks, or even method components [109]. These notions are competing and are not consensual: a method fragment can be either a process fragment, which is related to the activities performed on a method, or a product fragment, which represents the outcomes of the activities of a method; in turn, the most common definition of a method chunk is that it consists of a combination of a process and a product part and has a descriptor describing the context in which it can be used; finally, a method component is composed by method elements and their goals, being self-contained since it describes process for transforming input artifacts onto output artifacts and notation used for doing it.

Figure 47 depicts a high-level representation of the SME domain (Based on: [104]) of studying: how to create method fragments; how to store and retrieve method fragments; how to assemble a complete method from the fragments; and how to formalize the fragments. In SME, atomic components of methods are stored in a repository, also denominated method base, along with information on its characteristics or context of usage, so that they can be chosen and combined into a method according to the specific context of application.
(in this diagram, it is assumed that a *method chunk* is the combination of a *product fragment* plus one *process fragment*), a *process to construct* the methodology, and a *process for identification of method chunks and fragments* (either from existing methods or from scratch). *Method fragments* should conform to a *metaclass* belonging to a *methodology metamodel* (for instance, the ones presented in previous subsection).

Processes for identification of method chunks fragments are surveyed in [104]. It is possible to identify and define chunks from existing methods using: process or product-driven decomposition approaches, which modularizes a method, decomposing it into lower level constituents; through an exploratory approach, which explores several alternatives to a determined process model checking its adequacy to different situations; or through ad-hoc approach, which deals with the identification, evaluation, and refinement of chunks, in an iterative fashion, on the basis of experience and/or theory, until the chunk achieves the desired quality level [110].

The same survey also discusses processes for the creation of methods using chunks, either from scratch or using an existing method as source, using three approaches [111]: *assembly-based*, *extension-based*, and *paradigm-based*. The *assembly-based* strategy for the creation of method involves using method chunks stored in a *method base*. The *extension-based* approach involves the usage of patterns applied to existing methods. Finally, the *paradigm-based* approach involves instantiation or adaptation from a meta-model, or abstraction from an existing model. Processes for tailoring are also discussed in the same survey.

Industry uptake due to perceived high costs and the automation of the method construction process are seen as the main challenges to the adoption of these practices [104]. However, that perception is changing with initiatives such as Software Engineering Method and Theory (SEMAT) [9]. The SEMAT initiative aims to reinvent the software engineering practice based on theory and proven practices, gathering support from both industry and academia. At the date of this writings, a proposal involving SEMAT and OMG was created for a scalable and extensible kernel of software engineering domain concepts and relationships and a domain-specific language to allow the description of the practices by practitioners themselves, shifting the control from method engineers to developer teams.

### 6.3 Enterprise Architecture Management Patterns

In a similar fashion to that of method engineering, EA management patterns aim at adapting the process of developing and maintaining an EA to the real needs of organizations [89]. Organizations often make an option for established EA frameworks, which are too abstract or too extensive, often not taking advantage of related initiatives of frameworks existing inside of the organization. These approaches often result in a labor-intensive and time-consuming process, which is out of phase with the real needs and problems of organizations.

EA patterns document solutions to typical recurring problems, in the context of EA management, based on best-practices [90]. Patterns are reusable and dependent on the context of application, and are problem-driven, providing an entry-point for EA management activities. Three different types of patterns are considered: *Methodology Patterns* (M-Pattern), *Viewpoint Patterns* (V-Patterns), and *Information Model Patterns* (I-Patterns).

*M-Patterns* define activities and procedures that are carried out during specific points in the process of EA management. A given example is TOGAF, which provides ADM as a process model for managing EA, but leaves out details of the methods supporting the specific activities carried out during the ADM phases [89]. *V-Patterns* provide visualizations for information captured in the I-Patterns, in an adequate way for its purpose, and can be used by one or more M-Patterns. Example visualizations include diagrams, tables, and reports. *I-Patterns* provide information models based on best-practices and are used to define the kind of information that should be collected in EA management. *I-Patterns* can use formal languages like UML, or informal languages, such as natural language. A pattern catalog is currently available online [10].

Though a formal process for integrating patterns into a EA method/process is not defined, some guidelines are provided in [112] on the integration and on avoiding discrepancies/conflicts. The current focus of EA management is on providing patterns for general tasks of EA management activities [88]. Additionally, it is also suggested the use of patterns to complement already existing frameworks and processes [112].

### 6.4 Summary

The methods described in section 5 contrast with the situational method engineering approach also described in the current section. It is highly debatable...
whether which approach has more advantages: off-the-shelf methods already provide a process and guidelines but are highly generic and might be difficult to apply in determined contexts; on the other hand, situational methods can be more easily applied to determined situations but the engineering of a situational method is complex and has associated costs. In addition to this, there seems to be a lack of consensus on the terminology used in method engineering, with each specification offering its own take on the subject, which might add to the confusion of practitioners when the time of adopting a specification for doing real work.

7 Conclusions

This proposal aims to innovate on the methodological approaches to the engineering of systems that truly takes into account the context through the usage of the concept of capability as a means for characterizing and engineering systems in systems-of-systems contexts. For that purpose, the follow-up work will aim the definition of the concept of capability of a system along with the definition of architecture viewpoints to make it possible to describe systems and their components by the means of their engineered capabilities. Research on methods for capability-based engineering of systems, that are in contrast with traditional systems engineering methods is also to be pursued. Additionally, methods to combine systems (their capabilities) in order to deliver new types of capabilities will also be researched.

As preliminary work, we described the concepts which form the foundation of this research, in particular the concept of capability and the multiple domains where it was adopted and used. With basis on that analysis, a description of the different aspects of a capability was presented, combining knowledge from the different domains approached. A description of relevant modeling frameworks also encompassing different domains was also provided, along with an analysis of their coverage of the different aspects of a capability. Relevant methods for the engineering of systems from the domains of software engineering, enterprise architecture, and systems engineering was provided along with an analysis of their coverage of the system’s life cycle. Finally, a description of the method engineering practice including relevant process meta-models, situational method engineering, and enterprise architecture management patterns was provided.

A preliminary result of this work can be seen in [113], which looked into developing a reference architecture for digital preservation using a capability-based approach.

References


[60] EIA-731.1 - Systems Engineering Capability Model, Electronic Industries Alliance Std. EIA-731.1.


[65] Business Motivation Model 1.1, Object Management Group Std.


